Competitive Security Assessment

dappOS-Stader

Nov 16th, 2023

Secure3

secure3.io

Summary

This report is prepared for the project to identify vulnerabilities and issues in the smart contract source code. A group of NDA covered experienced security experts have participated in the Secure3's Audit Contest to find vulnerabilities and optimizations. Secure3 team has participated in the contest process as well to provide extra auditing coverage and scrutiny of the finding submissions.

The comprehensive examination and auditing scope includes:

- Cross checking contract implementation against functionalities described in the documents and white paper disclosed by the project owner.
- Contract Privilege Role Review to provide more clarity on smart contract roles and privilege.
- Using static analysis tools to analyze smart contracts against common known vulnerabilities patterns.
- Verify the code base is compliant with the most up-to-date industry standards and security best practices.
- Comprehensive line-by-line manual code review of the entire codebase by industry experts.

The security assessment resulted in findings that are categorized in four severity levels: Critical, Medium, Low, Informational. For each of the findings, the report has included recommendations of fix or mitigation for security and best practices.

Overview

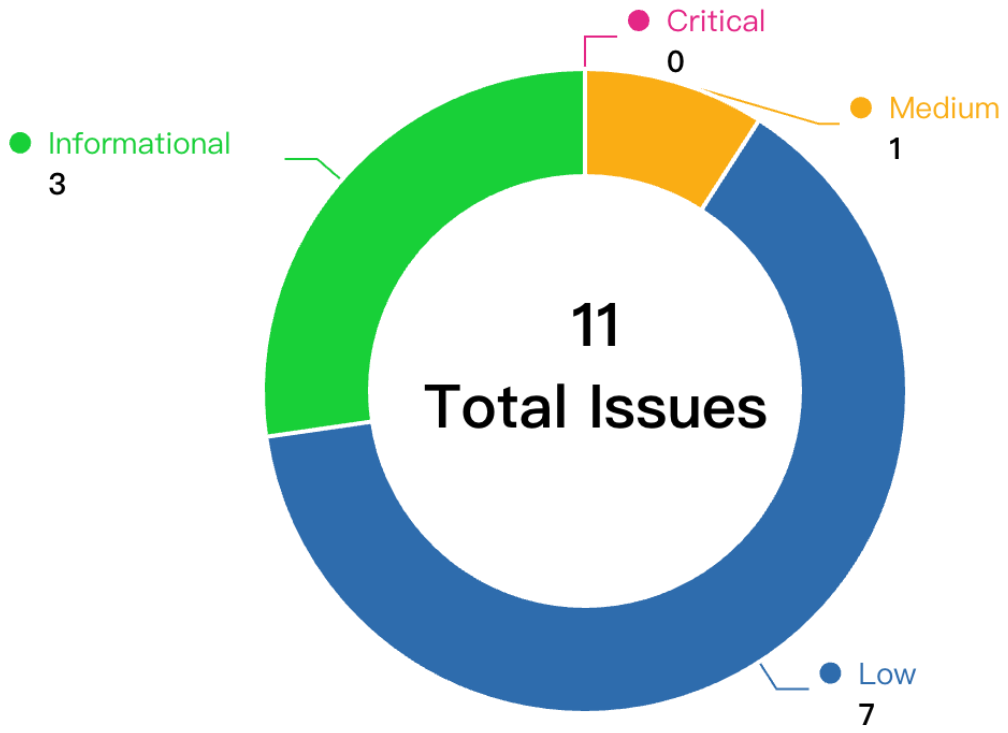
Project Detail

Project Name	dappOS-Stader
Platform & Language	Solidity
Codebase	• https://github.com/DappOSDao/ethx-convex-service • audit commit - 4ec6bef9a622edf81c71b96dcc3e571d3d9a978e • final commit - fba7e5198aa1011e567e7fb7aa62773704933b92
Audit Methodology	• Audit Contest • Business Logic and Code Review • Privileged Roles Review • Static Analysis

Audit Scope

File	SHA256 Hash
./contracts/StaderConvexService.sol	bee37d3333b0e38b7b1daa3907d9d8fcf3bea93fa61dd8264f8cb32acb4acfa8
./contracts/StaderService.sol	bbb53bb8493a764dd3a40e3c8bcac4580fb6ebd3b2681bb7e14e352464d3116d

Code Assessment Findings



ID	Name	Category	Severity	Client Response	Contributor
DAS-1	Missing slippage protection in StaderConvexService.sol	Logical	Medium	Fixed	comcat, Meliclit
DAS-2	ignore the reward tokens for Action.ethxLpWithdrawETHEOA	Logical	Low	Acknowledged	comcat
DAS-3	Adding liquidity to pools without gauge may be impossible	DOS	Low	Acknowledged	Meliclit
DAS-4	Contract address hardcoded	Code Style	Low	Acknowledged	yekong, Meliclit, Xi_Zi

DAS-5	add zero address check for the poolAdress and rewardPool	Logical	Low	Acknowledged	comcat
DAS-6	Reentrancy risk	Reentrancy	Low	Fixed	Xi_Zi
DAS-7	Refactoring Required for Complex Logic in execute Function of StaderService Contract	Code Style	Low	Fixed	yekong
DAS-8	Computational accuracy problem	Logical	Low	Fixed	Xi_Zi
DAS-9	The Action branch of an unhandled exception	Logical	Informational	Fixed	Xi_Zi
DAS-10	Redundant Operation in swapwETHToETH Function of StaderService Contract	Logical	Informational	Acknowledged	yekong
DAS-11	Redundant Event Declaration in StaderConvexService Contract	Code Style	Informational	Fixed	yekong

DAS-1:Missing slippage protection in StaderConvexService.sol

Category	Severity	Client Response	Contributor
Logical	Medium	Fixed	comcat, Meliclit

Code Reference

- code/contracts/StaderConvexService.sol#L213
- code/contracts/StaderConvexService.sol#L237

```
213:uint256 lpReceived = ICurveEthEthxPool(eTHxConvexContract.curveEthEthxPool).add_liquidity{value:
ethAmount}(amounts, 0);

213:uint256 lpReceived = ICurveEthEthxPool(eTHxConvexContract.curveEthEthxPool).add_liquidity{value:
ethAmount}(amounts, 0);

237:uint256 lpReceived = ICurvePool(eTHxConvexContract.poolAddress).add_liquidity(amounts, 0);
```

Description

comcat : inside the `StaderConvexService`, the function execute, when it handles the case for `Action(p.action) == Action.ethDeposit`, it will first deposit some ETH to gain ETHx, then add liquidity ETH and ETHx to the curve ETH_ETHx pool. however, when it add liquidity into the curve pool, it fails to consider the min return amount. which means that an MEV bot can arbitrage this tx and gain profit. since there is no min return check.
Meliclit : In StaderConvexService.sol, there exist multiple instances where 0 is passed as the minimum amount to the add_liquidity function. This means that there is no slippage protection, and users may lose their funds

```
StaderConvexService.sol
213: uint256 lpReceived = ICurveEthEthxPool(eTHxConvexContract.curveEthEthxPool).add_liquidity{valu
e: ethAmount}(amounts, 0);
```

Recommendation

comcat : add the corresponding min return check. maybe ask user to calculate it off chain and then pass the min return amount through params.
Meliclit : Allow the user to specify the minimum mint amount, like here

```
StaderConvexService.sol
159: uint256 lpReceived = ICurveEthEthxPool(eTHxConvexContract.curveEthEthxPool).add_liquidity{valu
e: withValue}(amounts, min_mint_amount);
```

Client Response

Fixed.

DAS-2:ignore the reward tokens for Action.ethxLpWithdrawETHEOA

Category	Severity	Client Response	Contributor
Logical	Low	Acknowledged	comcat

Code Reference

- code/contracts/StaderConvexService.sol#L395

```
395:} else if (Action(p.action) == Action.ethxLpWithdrawETHEOA) {
```

Description

comcat : for the `(Action(p.action) == Action.ethxLpWithdrawETHEOA)` condition, it will unstake the lp token from the reward pool, and remove liquidity one coin through the lp. however, it ignore the user's reward token. it should be swapped to the target token and transfer to receiver.

Recommendation

comcat : suggest use the same logic inside the `Action(p.action) == Action.withdrawOneCoin`, namely snapshot the reward token balance, and calculate the real amount received. and swap these tokens to ETH, and add to the amount user claimed.

Client Response

Acknowledged.The reward token we designed only stays in vw.

DAS-3:Adding liquidity to pools without gauge may be impossible

Category	Severity	Client Response	Contributor
DOS	Low	Acknowledged	Meliclit

Code Reference

- code/contracts/StaderConvexService.sol#L225
- code/contracts/StaderConvexService.sol#L243

```
225:eTHxConvexContract.rewardPool = ICurvePool(eTHxConvexContract.curveFactory).get_gauge(eTHxConvex
Contract.poolAddress);

243:IBooster(eTHxConvexContract.rewardPool).deposit(lpReceived);
```

Description

Meliclit : `get_gauge()` function returns address(0) if gauge doesn't exist

```
225: eTHxConvexContract.rewardPool = ICurvePool(eTHxConvexContract.curveFactory).get_gauge(eTHxConve
xContract.poolAddress);
```

The problem is that `get_gauge()` returned value is not checked and when liquidity is added to the pool `execute()` function tries to deposit lp tokens. This will lead to revert of transaction. Users will be unable to add liquidity to some pools

```
243: IBooster(eTHxConvexContract.rewardPool).deposit(lpReceived);
```

Recommendation

Meliclit : Add following check when Action.deposit and when Action.withdraw

```
if (eTHxConvexContract.rewardPool != address(0)) {
            if (IERC20(eTHxConvexContract.lpToken).allowance(address(this), eTHxConvexContract.rewar
dPool) == 0) {
                IERC20(eTHxConvexContract.lpToken).safeApprove(eTHxConvexContract.rewardPool, type(u
int).max);
            }

            IBooster(eTHxConvexContract.rewardPool).deposit(lpReceived);
}
```

Client Response

Acknowledged.

DAS-4:Contract address hardcoded

Category	Severity	Client Response	Contributor
Code Style	Low	Acknowledged	yekong, Meliclit, Xi_Zi

Code Reference

- code/contracts/StaderService.sol#L80-L83
- code/contracts/StaderService.sol#L81-L82
- code/contracts/StaderConvexService.sol#L95-L108
- code/contracts/StaderConvexService.sol#L100-L107
- code/contracts/StaderConvexService.sol#L451-L453

```
80:staderContract.wETH = 0xC02aaA39b223FE8D0A0e5C4F27eAD9083C756Cc2; //0xdf76b3c8088088E99388807f5fe2A4B3dF5D84fd; // 0xC02aaA39b223FE8D0A0e5C4F27eAD9083C756Cc2;
81:         staderContract.depositPool = 0xcf5EA1b38380f6aF39068375516Daf40Ed70D299; //0xd0e400Ec6Ed9C803A9D9D3a602494393E806F823; // 0xcf5EA1b38380f6aF39068375516Daf40Ed70D299;
82:         staderContract.withdrawManager = 0x9F0491B32DBce587c50c4C43AB303b06478193A7; //0x1048Eca024cB2Ba5eA720Ac057D804E95a809Fc8; // 0x9F0491B32DBce587c50c4C43AB303b06478193A7;
83:         staderContract.hybridPayService = 0x5E54182fa0d40F6954FA8BD9fEA0Ce639A32024f; //0x1B8d27E37cb431fb7D7210bDE5cAff0e52b6fc43; // 0x25b79Be1bb2c2e64f6c4C015c31aD570d954080F;

81:staderContract.depositPool = 0xcf5EA1b38380f6aF39068375516Daf40Ed70D299; //0xd0e400Ec6Ed9C803A9D9D3a602494393E806F823; // 0xcf5EA1b38380f6aF39068375516Daf40Ed70D299;
82:         staderContract.withdrawManager = 0x9F0491B32DBce587c50c4C43AB303b06478193A7; //0x1048Eca024cB2Ba5eA720Ac057D804E95a809Fc8; // 0x9F0491B32DBce587c50c4C43AB303b06478193A7;

95:ETHxConvexContractParam memory eTHxConvexContract;
96:         eTHxConvexContract.ETHx = 0xA35b1B31Ce002FBF2058D22F30f95D405200A15b;
97:         eTHxConvexContract.WETH = 0xC02aaA39b223FE8D0A0e5C4F27eAD9083C756Cc2;
98:         eTHxConvexContract.CRV = 0xD533a949740bb3306d119CC777fa900bA034cd52;
99:         eTHxConvexContract.CVX = 0x4e3FBD56CD56c3e72c1403e103b45Db9da5B9D2B;
100:         eTHxConvexContract.staderPool = 0xcf5EA1b38380f6aF39068375516Daf40Ed70D299;
101:         eTHxConvexContract.curveFactory = 0xF18056Bbd320E96A48e3Fbf8bC061322531aac99;
102:         eTHxConvexContract.curveEthEthxPoolId = 232;
103:         eTHxConvexContract.curveEthEthxPool = 0x59Ab5a5b5d617E478a2479B0cAD80DA7e2831492;
104:         eTHxConvexContract.curveWethEthxPool = 0xd82C2eB10F4895CABED6EDa6eeee234bd1A9838B;
105:         eTHxConvexContract.curveEthEthxRewardPool = 0x399e111c7209a741B06F8F86Ef0Fdd88fC198D20;
106:         eTHxConvexContract.convexBooster = 0xF403C135812408BFbE8713b5A23a04b3D48AAE31;
107:         eTHxConvexContract.withdrawManager = 0x9F0491B32DBce587c50c4C43AB303b06478193A7;
108:         eTHxConvexContract.hybridPayService = 0x5E54182fa0d40F6954FA8BD9fEA0Ce639A32024f;

100:eTHxConvexContract.staderPool = 0xcf5EA1b38380f6aF39068375516Daf40Ed70D299;
101:         eTHxConvexContract.curveFactory = 0xF18056Bbd320E96A48e3Fbf8bC061322531aac99;
102:         eTHxConvexContract.curveEthEthxPoolId = 232;
103:         eTHxConvexContract.curveEthEthxPool = 0x59Ab5a5b5d617E478a2479B0cAD80DA7e2831-
```

```
492;
104:        eTHxConvexContract.curveWethEthxPool = 0xd82C2eB10F4895CABED6EDa6eeee234bd1A9838B;
105:        eTHxConvexContract.curveEthEthxRewardPool = 0x399e111c7209a741B06F8F86Ef0Fdd88fC198D20;
106:        eTHxConvexContract.convexBooster = 0xF403C135812408BFbE8713b5A23a04b3D48AAE31;
107:        eTHxConvexContract.withdrawManager = 0x9F0491B32DBce587c50c4C43AB303b06478193A7;

451:uint256 staderPoolExchangeRate = IStaderDepositPool(0xcf5EA1b38380f6aF39068375516Daf40Ed70D299).
getExchangeRate();
452:        uint256 ethCurveEthEthxPoolAmount = ICurveEthEthxPool(0x59Ab5a5b5d617E478a2479B0cAD80DA7
e2831492).balances(0);
453:        uint256 ethxCurveEthEthxPoolAmount = ICurveEthEthxPool(0x59Ab5a5b5d617E478a2479B0cAD80DA
7e2831492).balances(1);
```

Description

yekong : The execute function in the StaderService contract currently contains hardcoded addresses for wETH, depositPool, withdrawManager, and hybridPayService. This approach poses several issues in terms of maintainability, readability, flexibility, and security. Hardcoding addresses directly in the contract's logic makes the contract less adaptable to changes and more prone to human errors, especially if these addresses need to be updated or are different across various networks (mainnet, testnet, etc.).

Meliclit : Both StaderConvexService.sol and StaderService.sol have hardcoded addresses for staderPool and withdrawManager

```
StaderService.sol
81: staderContract.depositPool = 0xcf5EA1b38380f6aF39068375516Daf40Ed70D299;
82: staderContract.withdrawManager = 0x9F0491B32DBce587c50c4C43AB303b06478193A7;
```

The issue arises because these contracts are upgradeable, and the correct addresses may be altered in the future by the Stader admin through StaderConfig.sol. If these addresses change, users may encounter difficulties in depositing or withdrawing their funds

```
StaderConfig.sol
235: function updateUserWithdrawManager(address _userWithdrawManager) external onlyRole(DEFAULT_ADMI
N_ROLE) {
236:         setContract(USER_WITHDRAW_MANAGER, _userWithdrawManager);
237: }
```

```
StaderConfig.sol
227: function updateStakePoolManager(address _stakePoolManager) external onlyRole(DEFAULT_ADMIN_ROL
E) {
228:         setContract(STAKE_POOL_MANAGER, _stakePoolManager);
229: }
```

Xi_Zi : Use hard-coded addresses in functions, such as:

```
eTHxConvexContract.ETHx = 0xA35b1B31Ce002FBF2058D22F30f95D405200A15b;
eTHxConvexContract.WETH = 0xC02aaA39b223FE8D0A0e5C4F27eAD9083C756Cc2;
//...
```

This way of hardcoding addresses is not flexible enough and can make contracts difficult to maintain. It is recommended that these addresses be passed as constructor arguments or otherwise configured.

Recommendation

yekong : Replace hardcoded addresses with contract-level constants or state variables. This can be achieved by defining these addresses as constants or by initializing them in the contract's constructor.

```
contract StaderService is IService {
    address public constant WETH_ADDRESS = 0xC02aaA39b223FE8D0A0e5C4F27eAD9083C756Cc2;
    ......
}
```

Meliclit : Use StaderConfig.sol to fetch the addresses of staderPool and withdrawManager. Here is a link that may help: https://github.com/stader-labs/ethx/blob/mainnet_V0/INTEGRATION.md
Xi_Zi : It is recommended that these addresses be passed as constructor arguments or otherwise configured.

Client Response

Acknowledged.The service contract needs to be stateless, so if the stager contract is upgraded we will redeploy the service contract.

DAS-5:add zero address check for the poolAdress and rewardPool

Category	Severity	Client Response	Contributor
Logical	Low	Acknowledged	comcat

Code Reference

- code/contracts/StaderConvexService.sol#L224-L226
- code/contracts/StaderConvexService.sol#L283-L285

```
224:eTHxConvexContract.poolAddress = ICurvePool(eTHxConvexContract.curveFactory).find_pool_for_coins
(tokens[0], tokens[1]);
225:            eTHxConvexContract.rewardPool = ICurvePool(eTHxConvexContract.curveFactory).get_gaug
e(eTHxConvexContract.poolAddress);
226:            eTHxConvexContract.lpToken = ICurvePool(eTHxConvexContract.curveFactory).get_token(e
THxConvexContract.poolAddress);

283:eTHxConvexContract.poolAddress = ICurvePool(eTHxConvexContract.curveFactory).find_pool_for_coins
(tokens[0], tokens[1]);
284:            eTHxConvexContract.rewardPool = ICurvePool(eTHxConvexContract.curveFactory).get_gaug
e(eTHxConvexContract.poolAddress);
285:            eTHxConvexContract.lpToken = ICurvePool(eTHxConvexContract.curveFactory).get_token(e
THxConvexContract.poolAddress);
```

Description

comcat : inside the `StaderConvexService`, for the condition `(Action(p.action) == Action.deposit)`, it requires user itself to pass in 2 token address, and then use these two token addresses to check on factory to gain the corresponding pool address. however, for the wrong tokens, the factory will just return address(0) instead of the correct address. but the logic failed to check the address(0)

Recommendation

comcat : add the corresponding checks, namely

```
require(eTHxConvexContract.poolAddress != address(0), "not listed pool");
require(eTHxConvexContract.rewardPool != address(0), "illegal");
require(eTHxConvexContract.lpToken != address(0), "illegal");
```

Client Response

Acknowledged.

DAS-6:Reentrancy risk

Category	Severity	Client Response	Contributor
Reentrancy	Low	Fixed	Xi_Zi

Code Reference

- code/contracts/StaderService.sol#L68-L72
- code/contracts/StaderConvexService.sol#L84-L88

```
68:function execute(
69:        uint code,
70:        bytes calldata data,
71:        address node
72:    ) external override returns (bool) {

84:function execute(
85:        uint code,
86:        bytes calldata data,
87:        address node
88:    ) external override returns (bool) {
```

Description

Xi_Zi : reentrancyGuard or other forms of reentrant attack protection are not used in the contract. When performing an external call in the execute function, you need to consider the possibility of a reentrant attack.

Recommendation

Xi_Zi : Reentrant protection is recommended for critical operations

Client Response

Fixed. We add the nonReentrant in execute function.

DAS-7:Refactoring Required for Complex Logic in execute Function of StaderService Contract

Category	Severity	Client Response	Contributor
Code Style	Low	Fixed	yekong

Code Reference

- code/contracts/StaderService.sol#L89-L122

```
89:if (Action(p.action) == Action.Deposit) {
90:            // swap weth to eth
91:            swapwETHToETH(staderContract.wETH, p.gasToken, p.remainGas);
92:            // decode params
93:            IStaderDepositPool.DepositParams memory params = abi.decode(data[32:], (IStaderDeposi
tPool.DepositParams));
94:
95:            // Deposit
96:            IStaderDepositPool(staderContract.depositPool).deposit{value: (params.depositAmount)}
(address(this));
97:
98:        } else if (Action(p.action) == Action.RequestWithdraw) {
99:            // decode params
100:            IStaderDepositPool.RequestWithdrawParams memory params = abi.decode(data[32:], (ISta
derDepositPool.RequestWithdrawParams));
101:            // UnDeposit
102:            if (IERC20(params.ethXAddress).allowance(address(this), staderContract.withdrawManag
er) == 0) {
103:                // approve
104:                IERC20(params.ethXAddress).approve(staderContract.withdrawManager, type(uint).ma
x);
105:            }
106:            IStaderDepositPool(staderContract.withdrawManager).requestWithdraw(params.ethXAmoun
t, address(this));
107:
108:        } else if (Action(p.action) == Action.Withdraw || Action(p.action) == Action.WithdrawEO
A) {
109:            // decode params
110:            IStaderDepositPool.ClaimParams memory params = abi.decode(data[32:], (IStaderDeposit
Pool.ClaimParams));
111:            bridgeParam.tokenABalanceBefore = uint128(address(this).balance);
112:            IStaderDepositPool(staderContract.withdrawManager).claim(params.requestId);
113:            // Withdraw current chain EOA
114:            if (params.receiver != address(0) && params.receiver != address(this)) {
115:                require(
116:                    params.receiver ==
117:                        IVWManager(IVirtualWalletV2(address(this)).vwm())
118:                            .walletOwner(address(this)),
119:                    "isolate receiver is not owner"
120:                );
121:                ERC20Helper.safeTransfer(address(0), params.receiver, uint128(address
```

```
(this).balance) - bridgeParam.tokenABalanceBefore);
122:            }
```

Description

yekong : The execute function within the StaderService contract contains complex and lengthy logic branches under multiple if and else if statements. Each branch handles distinct actions like Deposit, RequestWithdraw, and Withdraw (or WithdrawEOA). This structure leads to a bloated and less maintainable function, making it hard to understand and modify.

Recommendation

yekong : Refactor the execute function by extracting each action's logic into separate internal functions. This refactoring will enhance the readability and maintainability of the contract. It will also make the execute function more concise and focused on high-level logic flow.

```
function execute(...) external override returns (bool) {
    // ...other logic...

    if (Action(p.action) == Action.Deposit) {
        handleDeposit(data);
    } else if (Action(p.action) == Action.RequestWithdraw) {
        handleRequestWithdraw(data);
    } else if (Action(p.action) == Action.Withdraw || Action(p.action) == Action.WithdrawEOA) {
        handleWithdraw(data);
    }

    // ...other logic...
}

function handleDeposit(bytes calldata data) internal {
    // Deposit logic
    // ...
}

function handleRequestWithdraw(bytes calldata data) internal {
    // RequestWithdraw logic
    // ...
}

function handleWithdraw(bytes calldata data) internal {
    // Withdraw logic
    // ...
}
```

Client Response

Fixed.

DAS-8:Computational accuracy problem

Category	Severity	Client Response	Contributor
Logical	Low	Fixed	Xi_Zi

Code Reference

- code/contracts/StaderConvexService.sol#L204

```
204:uint256 ethAmount = ethStakeAmount * ethCurveEthEthxPoolAmount / (ethxCurveEthEthxPoolAmount * s
taderPoolExchangeRate / 1e18 + ethCurveEthEthxPoolAmount);
```

Description

Xi_Zi : In the ethDeposit action, the contract calculates the ethAmount by:

```
uint256 ethAmount = ethStakeAmount * ethCurveEthEthxPoolAmount / (ethxCurveEthEthxPoolAmount * stade
rPoolExchangeRate / 1e18 + ethCurveEthEthxPoolAmount);
```

Floating-point arithmetic is used in this calculation, and in Solidity, floating-point arithmetic is not secure and may result in loss of precision. Such calculations can make the value of ethAmount inaccurate, resulting in the ETH put into the staderPool not matching expectations.

Recommendation

Xi_Zi : To solve this problem, it is recommended to use integer arithmetic instead of floating-point arithmetic. Floating-point numbers can be converted to integers by multiplying both the numerator and denominator by a sufficiently large number. The calculation after repair is as follows:

```
uint256 ethAmount = ethStakeAmount * ethCurveEthEthxPoolAmount * 1e18 / (ethxCurveEthEthxPoolAmount
* staderPoolExchangeRate + ethCurveEthEthxPoolAmount * 1e18);
```

This fix will ensure that no loss of precision occurs in integer arithmetic, thus improving the accuracy of the calculation.

Client Response

Fixed.

DAS-9:The Action branch of an unhandled exception

Category	Severity	Client Response	Contributor
Logical	Informational	Fixed	Xi_Zi

Code Reference

- code/contracts/StaderService.sol#L89-L122

```
89:if (Action(p.action) == Action.Deposit) {
90:            // swap weth to eth
91:            swapwETHToETH(staderContract.wETH, p.gasToken, p.remainGas);
92:            // decode params
93:            IStaderDepositPool.DepositParams memory params = abi.decode(data[32:], (IStaderDeposi
tPool.DepositParams));
94:
95:            // Deposit
96:            IStaderDepositPool(staderContract.depositPool).deposit{value: (params.depositAmount)}
(address(this));
97:
98:        } else if (Action(p.action) == Action.RequestWithdraw) {
99:            // decode params
100:            IStaderDepositPool.RequestWithdrawParams memory params = abi.decode(data[32:], (ISta
derDepositPool.RequestWithdrawParams));
101:            // UnDeposit
102:            if (IERC20(params.ethXAddress).allowance(address(this), staderContract.withdrawManag
er) == 0) {
103:                // approve
104:                IERC20(params.ethXAddress).approve(staderContract.withdrawManager, type(uint).ma
x);
105:            }
106:            IStaderDepositPool(staderContract.withdrawManager).requestWithdraw(params.ethXAmoun
t, address(this));
107:
108:        } else if (Action(p.action) == Action.Withdraw || Action(p.action) == Action.WithdrawEO
A) {
109:            // decode params
110:            IStaderDepositPool.ClaimParams memory params = abi.decode(data[32:], (IStaderDeposit
Pool.ClaimParams));
111:            bridgeParam.tokenABalanceBefore = uint128(address(this).balance);
112:            IStaderDepositPool(staderContract.withdrawManager).claim(params.requestId);
113:            // Withdraw current chain EOA
114:            if (params.receiver != address(0) && params.receiver != address(this)) {
115:                require(
116:                    params.receiver ==
117:                        IVWManager(IVirtualWalletV2(address(this)).vwm())
118:                            .walletOwner(address(this)),
119:                    "isolate receiver is not owner"
120:                );
121:                ERC20Helper.safeTransfer(address(0), params.receiver, uint128(address
```

```
(this).balance) - bridgeParam.tokenABalanceBefore);
122:            }
```

Description

Xi_Zi : In the execute function, each branch of Action(p.action) is processed in detail, but unknown Action enumeration values are not processed. It is recommended to add a default error handling branch at the end of the execute function. To ensure that unknown actions do not cause the contract to behave unexpectedly.

Recommendation

Xi_Zi : It is recommended to add a default error handling branch at the end of the execute function.

Client Response

Fixed.

DAS-10:Redundant Operation in swapwETHToETH Function of StaderService Contract

Category	Severity	Client Response	Contributor
Logical	Informational	Acknowledged	yekong

Code Reference

- code/contracts/StaderService.sol#L62

```
62:require(wETHBalance >= remainGas, "gas not enough");
```

Description

yekong : In the swapwETHToETH function of the StaderService contract, there is an issue when wETHBalance is exactly equal to remainGas. In this scenario, the function deducts remainGas from wETHBalance and then calls IWNativeToken(wETH).withdraw(wETHBalance). However, after the deduction, wETHBalance becomes zero, making the withdraw call redundant. This operation does not convert any WETH to ETH as intended, potentially leading to unexpected outcomes in the contract's logic where ETH is expected.

Recommendation

yekong : Modify the swapwETHToETH function to include a condition that checks if wETHBalance is greater than remainGas before executing the withdraw operation. This ensures that the withdraw operation is only called when there is a meaningful amount of WETH to convert to ETH.

Client Response

Acknowledged.

DAS-11:Redundant Event Declaration in StaderConvexService Contract

Category	Severity	Client Response	Contributor
Code Style	Informational	Fixed	yekong

Code Reference

- code/contracts/StaderConvexService.sol#L27

```
27:event LogUint(uint256);
```

Description

yekong : In the StaderConvexService smart contract, the event LogUint(uint256) is declared but not used anywhere in the contract. This redundant event declaration contributes to unnecessary code clutter, which can potentially lead to confusion and maintenance issues. Removing unused code elements like this event declaration is a good practice to keep the codebase clean and maintainable.

Recommendation

yekong : To optimize the contract and enhance its maintainability, it is recommended to remove the LogUint(uint256) event declaration from the contract. This will help in reducing the overall code size and improving readability. Ensure that this event is indeed not required in any future logic of the contract before removing it.

Client Response

Fixed.

Disclaimer

This report is subject to the terms and conditions (including without limitation, description of services, confidentiality, disclaimer and limitation of liability) set forth in the Invoices, or the scope of services, and terms and conditions provided to you ("Customer" or the "Company") in connection with the Invoice. This report provided in connection with the services set forth in the Invoices shall be used by the Company only to the extent permitted under the terms and conditions set forth in the Invoice. This report may not be transmitted, disclosed, referred to or relied upon by any person for any purposes, nor may copies be delivered to any other person other than the Company, without Secure3's prior written consent in each instance.

This report is not an "endorsement" or "disapproval" of any particular project or team. This report is not an indication of the economics or value of any "product" or "asset" created by any team or project that contracts Secure3 to perform a security assessment. This report does not provide any warranty or guarantee of free of bug of codes analyzed, nor do they provide any indication of the technologies, business model or legal compliancy.

This report should not be used in any way to make decisions around investment or involvement with any particular project. Instead, it represents an extensive assessing process intending to help our customers increase the quality of their code and high-level consistency of implementation and business model, while reducing the risk presented by cryptographic tokens and blockchain technology.

Secure3's position on the final decisions over blockchain technologies and corresponding associated transactions is that each company and individual are responsible for their own due diligence and continuous security.

The assessment services provided by Secure3 is subject to dependencies and under continuing development. The assessment reports could include false positives, false negatives, and other unpredictable results. The services may access, and depend upon, multiple layers of third-parties.