Competitive Security Assessment

Shield Staking Vault

Feb 26th, 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	Shield Staking Vault
Platform & Language	Solidity
Codebase	• https://github.com/ShieldDAODev/shield-staking-vault-v1 • audit commit - 2a6bdc4a9de461b1e467788b6772b397b1fdaa56 • final commit - 1209280dce85ca8d549986405407da990e95c694
Audit Methodology	• Audit Contest • Business Logic and Code Review • Privileged Roles Review • Static Analysis

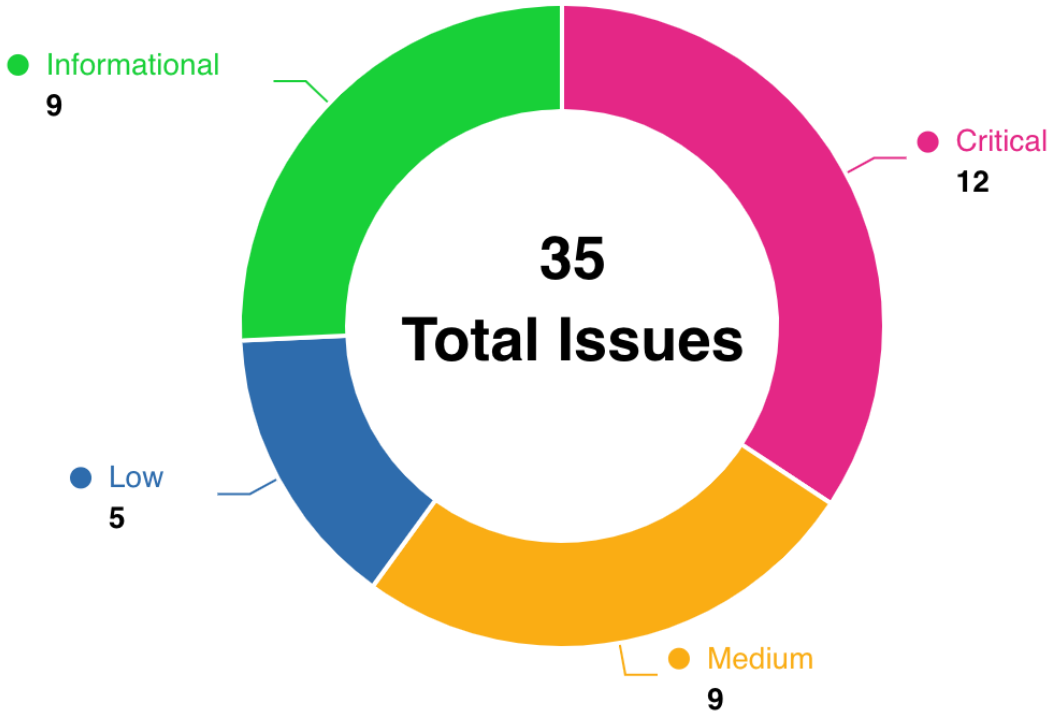
Code Vulnerability Review Summary

Vulnerability Level	Total	Reported	Acknowledged	Fixed	Mitigated	Declined
Critical	12	0	0	11	0	1
Medium	9	0	1	8	0	0
Low	5	0	0	5	0	0
Informational	9	0	2	6	0	1

Audit Scope

File	Commit Hash
contracts/interfaces/IAggregatorV3.sol	2a6bdc4a9de461b1e467788b6772b397b1fdaa56
contracts/interfaces/IBroker.sol	2a6bdc4a9de461b1e467788b6772b397b1fdaa56
contracts/interfaces/ICurveGauge.sol	2a6bdc4a9de461b1e467788b6772b397b1fdaa56
contracts/interfaces/IQuoter.sol	2a6bdc4a9de461b1e467788b6772b397b1fdaa56
contracts/interfaces/IStableSwap.sol	2a6bdc4a9de461b1e467788b6772b397b1fdaa56
contracts/libraries/Vault.sol	2a6bdc4a9de461b1e467788b6772b397b1fdaa56
contracts/libraries/VaultMath.sol	2a6bdc4a9de461b1e467788b6772b397b1fdaa56
contracts/OptionsTrading.sol	2a6bdc4a9de461b1e467788b6772b397b1fdaa56
contracts/PriceFeeder.sol	2a6bdc4a9de461b1e467788b6772b397b1fdaa56
contracts/StakingVault.sol	2a6bdc4a9de461b1e467788b6772b397b1fdaa56
contracts/structs/VaultStorage.sol	2a6bdc4a9de461b1e467788b6772b397b1fdaa56
contracts/VaultManager.sol	2a6bdc4a9de461b1e467788b6772b397b1fdaa56

Code Assessment Findings



ID	Name	Category	Severity	Status	Contributor
SSV-1	Dust LP will be left in the `StakingVault` contrat	Logical	Low	Fixed	jayphbee
SSV-2	Malicious collectShares operations on other users	Logical	Medium	Acknowledged	Kong7ych3
SSV-3	Risk of inability to modify Gov role leading to excessive authority	Privilege Related	Informational	Fixed	Kong7ych3
SSV-4	Users can get `StakingVault` shares for free	Logical	Critical	Declined	jayphbee

SSV-5	`OptionsTrading.latestRoundID` should be updated	Logical	Medium	Fixed	w2ning, jayphbee, Kong7ych3
SSV-6	`OptionsTrading::onlyGovernance` Wrong caller check	Logical	Critical	Fixed	Kong7ych3
SSV-7	`OptionsTrading::payOptionYield` redundant `msg.value` check	Code Style	Informational	Fixed	jayphbee, Kong7ych3
SSV-8	`StakingVault.SERIAL_NUMBER` is not guarenteed to be unique	Logical	Informational	Declined	jayphbee
SSV-9	`StakingVault._minEtherReceived` Incorrect slippage control	Logical	Critical	Fixed	thereksfour, Kong7ych3
SSV-10	`StakingVault.curveEnabled` is redundant	Gas Optimization	Informational	Fixed	alansh
SSV-11	`StakingVault.minDeposit` is not initialized	Logical	Low	Fixed	jayphbee
SSV-12	`StakingVault.withdrawableAmountInRound` is not used correctly	Logical	Critical	Fixed	thereksfour, alansh
SSV-13	`StakingVault::cancelWithdraw` should update `withdrawingSharesInRound` and `totalWithdrawingShares`	Logical	Critical	Fixed	thereksfour, alansh, Kong7ych3
SSV-14	`StakingVault::depositFor` can bypass the `maxVolume` limit	Logical	Medium	Fixed	thereksfour, w2ning, jayphbee, Kong7ych3
SSV-15	`StakingVault::deposit` wrong use of `msg.value` and `address.balance`	Logical	Medium	Fixed	comcat
SSV-16	`StakingVault::exit` Implementation not completed	Logical	Critical	Fixed	thereksfour, alansh, w2ning, comcat, jayphbee, Kong7ych3
SSV-17	`StakingVault::exit` always reverts	Logical	Critical	Fixed	thereksfour, jayphbee, Kong7ych3

SSV-18	`StakingVault::getEtherOnLPMining` Call nested in the staticcall can not modify state	Logical	Medium	Fixed	comcat
SSV-19	`StakingVault::getEtherOnLPMining` calculation optimization for `etherOut`	Gas Optimization	Informational	Fixed	alansh
SSV-20	`StakingVault::initialize` should check whether `endTime` is as expected	Logical	Low	Fixed	iczc, comcat, jayphbee, Kong7ych3
SSV-21	`StakingVault::initiateWithdraw` should increase `withdrawingSharesInRound`	Logical	Critical	Fixed	thereksfour
SSV-22	`StakingVault::sellAllLDO` Incorrect recipient address	Logical	Critical	Fixed	jayphbee, Kong7ych3
SSV-23	`StakingVault::sellAllLDO` `WETH` is not unwrapped	Logical	Critical	Fixed	jayphbee
SSV-24	`StakingVault::settlement` Anyone can provide `_minLPMint` and `_minEtherReceived` leading to sandwich attack	Race Condition	Medium	Fixed	thereksfour, jayphbee, Kong7ych3
SSV-25	`StakingVault::settlement` Incorrect handling of `etherNeedToSell`	Logical	Critical	Fixed	thereksfour
SSV-26	`StakingVault::settlement` The number of shares minted is excessive	Logical	Medium	Fixed	thereksfour
SSV-27	`StakingVault::settlement` does not send premium amount ether to `OptionsTrading`	Logical	Low	Fixed	jayphbee
SSV-28	`StakingVault::settlement` wrong parameter order when calling `VaultMath.getPremium`	Code Style	Low	Fixed	alansh, Kong7ych3
SSV-29	`StakingVault` Missing events for critical parameters	Code Style	Informational	Acknowledged	comcat
SSV-30	`StakingVault` Risk of price manipulation	Price Manipulation	Critical	Fixed	Kong7ych3

SSV-31	`StakingVault` does not work when `curveEnabled` is `false`	Logical	Medium	Fixed	alansh, comcat
SSV-32	`VaultManager` Gas optimization by using `immutable`	Gas Optimization	Informational	Fixed	comcat
SSV-33	`VaultManager` Missing events and caps for the `setManagementFeeRate` and `setPerformanceFeeRate`	Code Style	Informational	Acknowledged	comcat, jayphbee
SSV-34	`VaultStorage` Gas optimization by reorganizing storage layout	Gas Optimization	Informational	Fixed	comcat
SSV-35	`receive()` is not implemented in `OptionTrading` and `StakingVault` contract	Logical	Medium	Fixed	comcat, jayphbee

SSV-1:Dust LP will be left in the `StakingVault` contrat

Category	Severity	Code Reference	Status	Contributor
Logical	Low	• code/contracts/StakingVault.sol#L648-L651	Fixed	jayphbee

Code

```
648:         IStableSwap(STABLE_SWAP).remove_liquidity_imbalance(
649:             [_minEtherReceived.sub(sold), 0],
650:             lpAmount
651:         );
```

Description

jayphbee : As curve's doc says, `StableSwap.remove_liquidity_imbalance` returns actual amount of the LP tokens burned in the withdrawal. In the `StakingVault.terminate` function `remove_liquidity_imbalance` is called, but it's return value not be checked, so there will be dust LP left in the contract.

Recommendation

jayphbee : Burn all LP token using `StableSwap.remove_liquidity_one_coin` function.

Client Response

Fixed

SSV-2:Malicious collectShares operations on other users

Category	Severity	Code Reference	Status	Contributor
Logical	Medium	• code/contracts/StakingVault.sol#L331	Acknowledged	Kong7ych3

Code

```
331:    function collectShares(address _user) public notTerminated {
```

Description

Kong7ych3 : In the StakingVault contract, users can collect shares through the collectShares function. The number of shares depends on the roundPricePerShare parameter. In theory, the value of the roundPricePerShare parameter is different in each round. Therefore, it should be up to the user to decide whether to perform the collectShares operation in each round after depositing. However, arbitrary users can collect shares for other users through the collectShares function, which may confuse users who are forced to collect in unexpected roundPricePerShare situations.

Recommendation

Kong7ych3 : It is recommended to modify the collectShares function to internal visibility, and add an external visibility collectShares function, this external function can only collect msg.sender.

Client Response

This is by design. Later user can claim vault token during liquidity staking reward feature.

SSV-3:Risk of inability to modify Gov role leading to excessive authority

Category	Severity	Code Reference	Status	Contributor
Privilege Related	Informational	• code/contracts/OptionsTrading.sol#L54 • code/contracts/VaultManager.sol#L56	Fixed	Kong7ych3

Code

```
54:         governance = _governance;

56:         governance = msg.sender;
```

Description

Kong7ych3 : In the VaultManager contract, the governance role is set in the constructor, and there are no other functions in the contract that can modify the governance role. Such contracts are generally deployed for EOA, which will result in the governance authority being owned by the deployer EOA, and ownership transfer cannot be performed, which will lead to the risk of excessive governance role authority. The governance role in the OptionsTrading contract also has this problem.

Recommendation

Kong7ych3 : It is recommended to add a function to modify the governance role that can only be called by the governance role in the contract.

Client Response

Fixed

SSV-4:Users can get `StakingVault` shares for free

Category	Severity	Code Reference	Status	Contributor
Logical	Critical	• code/contracts/StakingVault.sol#L240 • code/contracts/StakingVault.sol#L256-L258 • code/contracts/StakingVault.sol#L294 • code/contracts/StakingVault.sol#L325	Declined	jayphbee

Code

```
240:            receipt.withdrawRound = latestRoundID;

256:            receipt.withdrawableAmount = receipt.withdrawableAmount.add(
257:                withdrawAmount
258:            );

294:        require(receipt.withdrawableAmount > 0, "no withdrawable");

325:            receipt.withdrawRound = 0;
```

Description

jayphbee : User can withdraw funds without burning his vault shares. Here's the proof of concept:

1. Alice deposits ether by calling `deposit`.
2. Alice initiates a withdraw by calling `initiateWithdraw`, the `else if` clause will be executed, `receipt.withdrawRound` will be updated to `latestRoundID`.
3. Alice calls `completeWithdraw` to finish the withdraw.
4. Alice deposits some ether by calling `deposit`.
5. Alice initiates a withdraw by calling `initiateWithdraw`, the `else` clause will be executed this time. `receipt.withdrawableAmount` is updated to a non zero value.
6. Alice calls `cancelWithdraw` to cancel **all** her shares, the corresponding vault shares is returned to Alice and `repceipt.withdrawRound` is set to 0.
7. Alice calls `completeWithdraw`, funds is transfer back to Alice.

We can see that Alice's funds are fully withdrawn but her vault share is not transfer to `StakingVault` contract. The impact is that Alice gets the StakingVault share for free. She can use the shares to withdarw funds by calling `StakingVault.exit` function when the vault is terminated.

Recommendation

jayphbee : Decrese the `receipt.withrawableAmount` accordingly when calling `cancelWithdraw`.

```
function cancelWithdraw(uint256 _shares)
        external
        nonReentrant
        notTerminated
    {
        require(_shares > 0, "less than zero");

        Vault.UserReceipt storage receipt = userReceipts[msg.sender];
        require(
            receipt.withdrawRound == latestRoundID,
            "no scheduled withdrawal"
        );
        require(receipt.withdrawShares >= _shares, "not enough");

        receipt.withdrawShares = receipt.withdrawShares.sub(_shares);
        _transfer(address(this), msg.sender, _shares);

        if (receipt.withdrawShares == 0) {
            receipt.withdrawRound = 0;
        }

        uint256 withdrawAmount = VaultMath.sharesToAsset(
            _shares,
            roundPricePerShare[receipt.withdrawRound]
        );
        receipt.withdrawableAmount = receipt.withdrawableAmount.sub(withdrawAmount);

        emit CancelWithdraw(msg.sender, _shares, latestRoundID);
    }
```

Client Response

Declined, this is not an issue.

SSV-5: `OptionsTrading.latestRoundID` should be updated

Category	Severity	Code Reference	Status	Contributor
Logical	Medium	• code/contracts/OptionsTrading.sol #L17	Fixed	w2ning, jayphbee, Kong7ych3

Code

```
17:    uint256 public latestRoundID;
```

Description

w2ning : OptionsTrading.latestRoundID (code/contracts/OptionsTrading.sol#17) can never be changed.

The impact is that the malicious contract can cause following functions in the contract to not work properly:

- OptionsTrading.tradeOptions
- OptionsTrading.payOptionYield
- OptionsTrading.getBalance
- OptionsTrading.refreshBalance

jayphbee : There's no code to update `latestRoundID` in `OptionsTrading` contract. It remains the default 0 value all the time, which will lead to the options trading details incorrectly recorded. Further more stale data will be returned in `OptionsTrading.getBalance` and `OptionsTrading.refreshBalance` functions.

Kong7ych3 : In the OptionsTrading contract, the StakingVault contract can obtain the proceeds of options trading through the roll function. It will reset the account of each round through `optionsAccounts[_round] = account`. But the latestRoundID variable is not reset to `_round`. This will cause latestRoundID to always be 0, causing tradeOptions, payOptionYield, getBalance and refreshBalance functions to fail to work normally.

Recommendation

w2ning : Change the value of `latestRoundID` in the `roll` function

Consider below fix in the `OptionsTrading.roll()` function

```
    function roll(uint256 _round, uint256 _premium)
        external
        payable
        onlyVault
        returns (uint256 tranferAmount)
    {

        // fix: Change the value of latestRoundID first
        latestRoundID = _round;

        OptionsAccount memory account;
        account.totalAllocation = _premium;
        account.balance = _premium;

        account.optionsBoughtAmount = 0;
        account.optionsYieldReturned = 0;
        account.lastOptionPurchaseTime = 0;
        account.shouldPayback = false;

        optionsAccounts[_round] = account;

        if (address(this).balance > _premium) {
            TransferHelper.safeTransferETH(
                vault,
                address(this).balance.sub(_premium)
            );
        } else {
            tranferAmount = _premium.sub(address(this).balance);
        }
    }
```

jayphbee : update `latestRoundID` in the `roll` function.

```
latestRoundID = _round
```

Kong7ych3 : It is recommended to perform the `latestRoundID` setting operation in the roll function.

Client Response

Fixed

SSV-6: `OptionsTrading::onlyGovernance` Wrong caller check

Category	Severity	Code Reference	Status	Contributor
Logical	Critical	• code/contracts/OptionsTrading.sol #L47	Fixed	Kong7ych3

Code

```
47:         require(msg.sender == vault, "not governance");
```

Description

Kong7ych3 : In the OptionsTrading contract, the onlyGovernance modifier is used to check if the caller is a governance role, but it incorrectly checks that the caller must be a vault contract. This causes functions in the OptionsTrading contract that should have been called by the governance role to never be used.

Recommendation

Kong7ych3 : It is recommended to modify the onlyGovernance decorator to: check that the caller is the governance role.

Client Response

Fixed

SSV-7: `OptionsTrading::payOptionYield` redundant `msg.value` check

Category	Severity	Code Reference	Status	Contributor
Code Style	Informational	• code/contracts/OptionsTrading.sol #L79-L81	Fixed	jayphbee, Kong7ych3

Code

```
79:         require(msg.value > 0, "no yields");
80:
81:         if (msg.value > 0) {
```

Description

jayphbee : Redundant check for `msg.value >` in `OptionsTrading.payOptionYield` function.

```
        require(msg.value > 0, "no yields");

        if (msg.value > 0) {
            account.balance = account.balance.add(msg.value);
            account.shouldPayback = false;
            account.optionsYieldReturned = account.optionsYieldReturned.add(
                msg.value
            );
        }
```

Kong7ych3 : In the OptionsTrading contract, the OptionsTrader role can pay option yield through the payOptionYield function. It first uses `require` to check that `msg.value` must be greater than 0, and then uses `if` to check that `msg.value > 0`. These two repeated checks are unnecessary.

Recommendation

jayphbee : Remove the `if (msg.value > 0)` check.
Kong7ych3 : It is recommended to remove the if condition of `msg.value > 0`.

Client Response

Fixed

SSV-8: `StakingVault.SERIAL_NUMBER` is not guarenteed to be unique

Category	Severity	Code Reference	Status	Contributor
Logical	Informational	• code/contracts/StakingVault.sol#L69	Declined	jayphbee

Code

```
69:         SERIAL_NUMBER = sha256(abi.encodePacked(msg.sender, block.timestamp));
```

Description

jayphbee : `SERIAL_NUMBER` is the identifier of a staking vault. It is derived from `msg.sender` and `block.timestamp`.

```
SERIAL_NUMBER = sha256(abi.encodePacked(msg.sender, block.timestamp));
```

But it isn't guarenteed to be unique when create StakingVault using `VaultManager.batchCreate` function, because `msg.sender` and `block.timestamp` is the same within the same transaction.

Recommendation

jayphbee : Add a `_counter` to the constructor of `StakingVault`, and derive SERIAL_NUMBER like:

```
SERIAL_NUMBER = sha256(abi.encodePacked(msg.sender, block.timestamp, _counter));
```

Client Response

`SERIAL_NUMBER` is used by the frontend dApp to identify if vaults are the 'same', which is defined as same address at the same tme.

SSV-9: `StakingVault._minEtherReceived` Incorrect slippage control

Category	Severity	Code Reference	Status	Contributor
Logical	Critical	• code/contracts/StakingVault.sol#L432-L435 • code/contracts/StakingVault.sol#L653-L656	Fixed	thereksfour, Kong7ych3

Code

```
432:             require(
433:                 _minEtherReceived.add(etherBefore) >= address(this).balance,
434:                 "slippage"
435:             );

653:         require(
654:             _minEtherReceived.add(etherBefore) >= address(this).balance,
655:             "slippage"
656:         );
```

Description

thereksfour : Use _minEtherReceived to apply slippage control to the amount of ETH received in settlement and terminate. However, due to a coding error, >= is used instead of <= to check the slippage, resulting in an invalid slippage check. In the following check, the ETH received by the contract is required to be less than _minEtherReceived

```
            require(
                _minEtherReceived.add(etherBefore) >= address(this).balance,
                "slippage"
            );
```

Kong7ych3 : In the StakingVault contract, the terminate function is used to stop the protocol from running. It will first withdraw LP tokens from CURVE_GAUGE, then remove liquidity through Curve Pool to obtain native tokens, and then perform a slippage check. The slippage check depends on the `_minEtherReceived` parameter passed in by the caller: `_minEtherReceived.add(etherBefore) >= address(this).balance`.

But it should be noted that `_minEtherReceived.add(etherBefore)` represents the minimum value of the expected number of native tokens accepted by the StakingVault contract, so after the liquidity is removed, the balance of native tokens in the StakingVault contract is greater than the minimum expected value is correct.

However, the actual implementation incorrectly checks that the minimum expected value must be greater than the native token balance in the contract, which will cause the terminate operation to suffer from a sandwich attack or the terminate operation may not be executed successfully.

The same goes for the slippage check in the settlement function.

Recommendation

thereksfour : Change to

```
            require(
-               _minEtherReceived.add(etherBefore) >= address(this).balance,
+               _minEtherReceived.add(etherBefore) <= address(this).balance,
                "slippage"
            );
```

Kong7ych3 : It is recommended to modify the slippage check to: `_minEtherReceived.add(etherBefore) <= address(this).balance`

Client Response

Fixed

SSV-10: `StakingVault.curveEnabled` is redundant

Category	Severity	Code Reference	Status	Contributor
Gas Optimization	Informational	• code/contracts/structs/VaultStorage.sol#L68-L70 • code/contracts/StakingVault.sol#L81-L84	Fixed	alansh

Code

```
68:     bool internal curveEnabled;
69:
70:     bool internal lpMiningEnabled;

81:         if (_curveEnabled) {
82:             curveEnabled = _curveEnabled;
83:             lpMiningEnabled = true;
84:         }
```

Description

alansh : `curveEnabled` is always assigned `true` together with `lpMiningEnabled = true;`, and never used later.So this variable can be removed and only `lpMiningEnabled` is needed.

Recommendation

alansh : Remove the `curveEnabled` variable from `VaultStorage`, and change function parameter `_curveEnabled` to `_lpMiningEnabled` to make the naming convention consistent.

Client Response

Fixed

SSV-11: `StakingVault.minDeposit` is not initialized

Category	Severity	Code Reference	Status	Contributor
Logical	Low	• code/contracts/StakingVault.sol#L160	Fixed	jayphbee

Code

```
160:         require(_amount > minDeposit, "too small");
```

Description

jayphbee : `minDesposit` storage variable is not initialized in StakingVault contract thus 1 wei amount of ether can be deposited.

```
    function _depositFor(uint256 _amount, address _user) internal {
        require(_amount > minDeposit, "too small");
```

Recommendation

jayphbee : Initialize `minDesposit` to a minimum value that a user can deposit in the `initialize` function.

```
minDesposit = _minDesposit;
```

Client Response

Fixed

SSV-12: `StakingVault.withdrawableAmountInRound` is not used correctly

Category	Severity	Code Reference	Status	Contributor
Logical	Critical	• code/contracts/StakingVault.sol#L243-L260 • code/contracts/StakingVault.sol#L514-L528	Fixed	thereksfour, alansh

Code

```
243:         } else {
244:             // Withdraw previous round share first
245:             uint256 withdrawAmount = VaultMath.sharesToAsset(
246:                 receipt.withdrawShares,
247:                 roundPricePerShare[receipt.withdrawRound]
248:             );
249:
250:             _burn(address(this), receipt.withdrawShares);
251:
252:             totalWithdrawingShares = totalWithdrawingShares
253:                 .sub(receipt.withdrawShares)
254:                 .add(_shares);
255:             receipt.withdrawShares = _shares;
256:             receipt.withdrawableAmount = receipt.withdrawableAmount.add(
257:                 withdrawAmount
258:             );
259:             receipt.withdrawRound = latestRoundID;
260:         }

514:     function currentSharePrice() public view returns (uint256) {
515:         if (totalSupply() == 0) {
516:             return MULTIPLIER;
517:         }
518:
519:         (, , uint256 totalFee) = getFees();
520:
521:         uint256 etherAmount = getAllEtherValue()
522:             .sub(totalPendingAmount)
523:             .sub(withdrawableAmountInRound)
524:             .sub(totalFee);
525:         uint256 shareAmount = totalSupply().sub(withdrawingSharesInRound);
526:
527:         return etherAmount.mul(MULTIPLIER).div(shareAmount);
528:     }
```

Description

thereksfour : currentSharePrice() is used to determine the price of the shares for the current round, where withdrawableAmountInRound will be used to calculate etherAmount

```
    function currentSharePrice() public view returns (uint256) {
        if (totalSupply() == 0) {
            return MULTIPLIER;
        }

        (, , uint256 totalFee) = getFees();

        uint256 etherAmount = getAllEtherValue()
            .sub(totalPendingAmount)
            .sub(withdrawableAmountInRound)
            .sub(totalFee);
        uint256 shareAmount = totalSupply().sub(withdrawingSharesInRound);

        return etherAmount.mul(MULTIPLIER).div(shareAmount);
    }
```

withdrawableAmountInRound is used to indicate the amount of ETH that the user is going to withdraw, but has not yet withdrawn This happens in the initiateWithdraw function, when withdrawRound < latestRoundID && withdrawRound ! = 0, the user will first save the amount of ETH to be taken out in withdrawableAmount and then take it out in completeWithdraw(), but withdrawableAmountInRound is not used correctly in this process

```
        } else {
            // Withdraw previous round share first
            uint256 withdrawAmount = VaultMath.sharesToAsset(
                receipt.withdrawShares,
                roundPricePerShare[receipt.withdrawRound]
            );

            _burn(address(this), receipt.withdrawShares);

            totalWithdrawingShares = totalWithdrawingShares
                .sub(receipt.withdrawShares)
                .add(_shares);
            receipt.withdrawShares = _shares;
            receipt.withdrawableAmount = receipt.withdrawableAmount.add(
                withdrawAmount
            );
            receipt.withdrawRound = latestRoundID;
        }
```

Since withdrawableAmountInRound is 0, the etherAmount in currentSharePrice() will be large, resulting in a large result in currentSharePrice() and thus a large roundPricePerShare.

alansh : `withdrawableAmountInRound` is logically supposed to be updated together with `withdrawingSharesInRound`, currently it is never updated and always `0`. For gas optimization, this variable can be

removed and calculate from `withdrawingSharesInRound` when needed. Thus user gas will be reduced, and the calculation is only triggered when settlement, which is much less frequent.

Recommendation

thereksfour : There are two solutions, one is to increase withdrawableAmountInRound in initiateWithdraw and decrease withdrawableAmountInRound in completeWithdraw. The other is to send ETH directly to the user in initiateWithdraw, so that you don't need to use withdrawableAmountInRound.
alansh : The calculation is:

```
withdrawableAmountInRound = VaultMath.sharesToAsset(withdrawingSharesInRound, sharePrice)
```

But there's a circular dependency as calculation of `sharePrice` also depends on `withdrawableAmountInRound`.There should be some spec change in order to fix this circular dependency issue.

Client Response

Replaced `withdrawableAmountInRound` with a new variable called `withdrawableAmountInPast` and fixed the issue with the new logic.

SSV-13: `StakingVault::cancelWithdraw` should update `withdrawingSharesInRound` and `totalWithdrawingShares`

Category	Severity	Code Reference	Status	Contributor
Logical	Critical	• code/contracts/StakingVault.sol#L234-L243 • code/contracts/StakingVault.sol#L307-L329	Fixed	thereksfour, alansh, Kong7ych3

Code

```
234:         if (receipt.withdrawRound == latestRoundID) {
235:             receipt.withdrawShares = receipt.withdrawShares.add(_shares);
236:             totalWithdrawingShares = totalWithdrawingShares.add(_shares);
237:             withdrawingSharesInRound = withdrawingSharesInRound.add(_shares);
238:         } else if (receipt.withdrawRound == 0) {
239:             receipt.withdrawShares = _shares;
240:             receipt.withdrawRound = latestRoundID;
241:             totalWithdrawingShares = totalWithdrawingShares.add(_shares);
242:             withdrawingSharesInRound = withdrawingSharesInRound.add(_shares);
243:         } else {

307:     function cancelWithdraw(uint256 _shares)
308:         external
309:         nonReentrant
310:         notTerminated
311:     {
312:         require(_shares > 0, "less than zero");
313:
314:         Vault.UserReceipt storage receipt = userReceipts[msg.sender];
315:         require(
316:             receipt.withdrawRound == latestRoundID,
317:             "no scheduled withdrawal"
318:         );
319:         require(receipt.withdrawShares >= _shares, "not enough");
320:
321:         receipt.withdrawShares = receipt.withdrawShares.sub(_shares);
322:         _transfer(address(this), msg.sender, _shares);
323:
324:         if (receipt.withdrawShares == 0) {
325:             receipt.withdrawRound = 0;
326:         }
327:
328:         emit CancelWithdraw(msg.sender, _shares, latestRoundID);
329:     }
```

Description

thereksfour : In initiateWithdraw, when the user deposits shares, totalWithdrawingShares and withdrawingSharesInRound are increased, withdrawingSharesInRound is used in the currentSharePrice() to calculate roundPricePerShare.

```
    function initiateWithdraw(uint256 _shares)
        external
        nonReentrant
        notTerminated
    {
        require(_shares > 0, "less than zero");

        Vault.UserReceipt storage receipt = userReceipts[msg.sender];

        // Collect all shares
        if (receipt.pendingAmount > 0 || receipt.unredeemedShares > 0) {
            collectShares(msg.sender);
        }

        require(balanceOf(msg.sender) >= _shares, "exceed");

        if (receipt.withdrawRound == latestRoundID) {
            receipt.withdrawShares = receipt.withdrawShares.add(_shares);
            totalWithdrawingShares = totalWithdrawingShares.add(_shares);
            withdrawingSharesInRound = withdrawingSharesInRound.add(_shares);
```

In cancelWithdraw, the previously deposited shares are sent to the user, but the withdrawingSharesInRound and totalWithdrawingShares are not reduced here.

```
    function cancelWithdraw(uint256 _shares)
        external
        nonReentrant
        notTerminated
    {
        require(_shares > 0, "less than zero");

        Vault.UserReceipt storage receipt = userReceipts[msg.sender];
        require(
            receipt.withdrawRound == latestRoundID,
            "no scheduled withdrawal"
        );
        require(receipt.withdrawShares >= _shares, "not enough");

        receipt.withdrawShares = receipt.withdrawShares.sub(_shares);
        _transfer(address(this), msg.sender, _shares);

        if (receipt.withdrawShares == 0) {
            receipt.withdrawRound = 0;
        }

        emit CancelWithdraw(msg.sender, _shares, latestRoundID);
    }
```

Since withdrawingSharesInRound is large, shareAmount will be small in currentSharePrice(), resulting in a large result in currentSharePrice(), and thus roundPricePerShare will be large.

```
  function currentSharePrice() public view returns (uint256) {
        if (totalSupply() == 0) {
            return MULTIPLIER;
        }

        (, , uint256 totalFee) = getFees();

        uint256 etherAmount = getAllEtherValue()
            .sub(totalPendingAmount)
            .sub(withdrawableAmountInRound)
            .sub(totalFee);
        uint256 shareAmount = totalSupply().sub(withdrawingSharesInRound);

        return etherAmount.mul(MULTIPLIER).div(shareAmount);
    }
```

alansh : `withdrawingSharesInRound` records the amount of withdrawing shares of current round, so when cancel, should also update it.

Kong7ych3 : In the StakingVault contract, users can request withdrawal through initiateWithdraw, which will update the `receipt.withdrawShares`, `totalWithdrawingShares` and `withdrawingSharesInRound` parameters. Afterwards, the user can cancel the withdrawal request through the cancelWithdraw function, but the `totalWithdrawingShares` and `withdrawingSharesInRound` parameters are not updated in the cancelWithdraw function, which will lead to deviations in the accounting of deposits and withdrawals, and the `withdrawingSharesInRound` parameter will directly affect settlement operation.

Recommendation

thereksfour : Change to

```
    function cancelWithdraw(uint256 _shares)
        external
        nonReentrant
        notTerminated
    {
        require(_shares > 0, "less than zero");

        Vault.UserReceipt storage receipt = userReceipts[msg.sender];
        require(
            receipt.withdrawRound == latestRoundID,
            "no scheduled withdrawal"
        );
        require(receipt.withdrawShares >= _shares, "not enough");

        receipt.withdrawShares = receipt.withdrawShares.sub(_shares);
+       totalWithdrawingShares = totalWithdrawingShares.sub(_shares);
+        withdrawingSharesInRound = withdrawingSharesInRound.sub(_shares);
        _transfer(address(this), msg.sender, _shares);

        if (receipt.withdrawShares == 0) {
            receipt.withdrawRound = 0;
        }

        emit CancelWithdraw(msg.sender, _shares, latestRoundID);
    }
```

alansh : Consider below fix in the `StakingVault.cancelWithdraw()` function

```
receipt.withdrawShares = receipt.withdrawShares.sub(_shares);
withdrawingSharesInRound = withdrawingSharesInRound.sub(_shares);
totalWithdrawingShares = totalWithdrawingShares.sub(_shares);
```

Kong7ych3 : It is recommended to update the totalWithdrawingShares and withdrawingSharesInRound variables during the cancelWithdraw operation.

Client Response

Fixed

SSV-14: `StakingVault::depositFor` can bypass the `maxVolume` limit

Category	Severity	Code Reference	Status	Contributor
Logical	Medium	• code/contracts/StakingVault.sol#L133-L158 • code/contracts/StakingVault.sol#L149-L158	Fixed	thereksfour, w2ning, jayphbee, Kong7ych3

Code

```
133:     function deposit(address _broker)
134:         external
135:         payable
136:         nonReentrant
137:         notTerminated
138:     {
139:         Vault.RoundInfo memory round = roundInfo[latestRoundID];
140:
141:         require(msg.value.add(getAllEtherValue()) <= round.maxVolume, "exceed");
142:         _depositFor(msg.value, msg.sender);
143:
144:         if (broker != address(0) && _broker != address(0)) {
145:             IBroker(broker).addBrokerRelationship(_broker, msg.sender);
146:         }
147:     }
148:
149:     function depositFor(address _user)
150:         external
151:         payable
152:         nonReentrant
153:         notTerminated
154:     {
155:         require(_user != address(0), "ZERO ADDRESS");
156:         _depositFor(msg.value, _user);
157:     }
158:
```

```
149:     function depositFor(address _user)
150:         external
151:         payable
152:         nonReentrant
153:         notTerminated
154:     {
155:         require(_user != address(0), "ZERO ADDRESS");
156:         _depositFor(msg.value, _user);
157:     }
158:
```

Description

thereksfour : In StakingVault, users can deposit ETH via deposit or depositFor. In deposit, the volume is limited to maxVolume, but depositFor does not have this limit, so users can deposit ETH over maxVolume via depositFor.

```
    function deposit(address _broker)
        external
        payable
        nonReentrant
        notTerminated
    {
        Vault.RoundInfo memory round = roundInfo[latestRoundID];

        require(msg.value.add(getAllEtherValue()) <= round.maxVolume, "exceed");
        _depositFor(msg.value, msg.sender);

        if (broker != address(0) && _broker != address(0)) {
            IBroker(broker).addBrokerRelationship(_broker, msg.sender);
        }
    }

    function depositFor(address _user)
        external
        payable
        nonReentrant
        notTerminated
    {
        require(_user != address(0), "ZERO ADDRESS");
        _depositFor(msg.value, _user);
    }
```

w2ning : In `deposit` function, there is a check for maxVolume But in the `depositFor` function, this check is missing

```
function deposit(address _broker)
    external
    payable
    nonReentrant
    notTerminated
{
    Vault.RoundInfo memory round = roundInfo[latestRoundID];

    // There is a check for maxVolume
    require(msg.value.add(getAllEtherValue()) <= round.maxVolume, "exceed");
    _depositFor(msg.value, msg.sender);

    if (broker != address(0) && _broker != address(0)) {
        IBroker(broker).addBrokerRelationship(_broker, msg.sender);
    }
}

function depositFor(address _user)
    external
    payable
    nonReentrant
    notTerminated
{

    // Missing the check for maxVolume
    require(_user != address(0), "ZERO ADDRESS");
    _depositFor(msg.value, _user);
}
```

jayphbee : There is a hard limit that users can deposit in every round, but this limit isn't checked in the `depsoitFor` function. It should be checked like `deposit` function do.

Kong7ych3 : In the StakingVault contract, users can deposit native tokens through the deposit function. The deposit function will first check whether the total deposit in the contract is less than maxVolume, then call the `_depositFor` function to deposit for msg.sender, and finally perform the addBrokerRelationship operation. Users can also make deposits for specified users through the depositFor function, which will directly call the `_depositFor` function to make deposits for the specified users passed in by msg.sender, without performing the maxVolume check and addBrokerRelationship operation. This conflicts with the business logic of the deposit function. Users can deposit for themselves through the depositFor function to bypass the maxVolume check. And the user deposits through the depositFor function but lacks the addBrokerRelationship operation.

Recommendation

thereksfour : Change to

```
    function depositFor(address _user)
        external
        payable
        nonReentrant
        notTerminated
    {
        require(_user != address(0), "ZERO ADDRESS");
+       require(msg.value.add(getAllEtherValue()) <= round.maxVolume, "exceed");
        _depositFor(msg.value, _user);
    }
```

w2ning : Add the same check in the `depositFor` function.

Consider below fix in the `StakingVault.depositFor()` function

```
    function depositFor(address _user)
        external
        payable
        nonReentrant
        notTerminated
    {
        require(_user != address(0), "ZERO ADDRESS");
        // Add the same check
        require(msg.value.add(getAllEtherValue()) <= round.maxVolume, "exceed");
        _depositFor(msg.value, _user);
    }
```

jayphbee : check `maxVolume` like `deposit` function do.

```
    function depositFor(address _user)
        external
        payable
        nonReentrant
        notTerminated
    {
        require(_user != address(0), "ZERO ADDRESS");
        Vault.RoundInfo memory round = roundInfo[latestRoundID];

        require(msg.value.add(getAllEtherValue()) <= round.maxVolume, "exceed");
        _depositFor(msg.value, _user);
    }
```

Kong7ych3 : It is recommended to check maxVolume and perform addBrokerRelationship operation in the depositFor function.

Client Response

Fixed

SSV-15: `StakingVault::deposit` wrong use of `msg.value` and `address.balance`

Category	Severity	Code Reference	Status	Contributor
Logical	Medium	• code/contracts/StakingVault.sol#L141	Fixed	comcat

Code

```
141:        require(msg.value.add(getAllEtherValue()) <= round.maxVolume, "exceed");
```

Description

comcat : in the staking vault contract, the function `deposit`, whose inside check the `require(msg.value.add(getAllEtherValue()) <= round.maxVolume, "exceed");` it suppose to check wether it will exceed the deposit limit, however, it misused the `msg.value` and `address(this).balance`. for the payable function, `msg.value` will keep still, and `address(this).balance` will add the `msg.value` at the very first step.

so when i create a vault, and set the deposit limit to 1 ether. when i call the deposit function with msg.value = 1 ether, the requirement will fail with "exceed".

the reason is that: the function `getAllEtherVaule()` calculate the `value = valueInvested + address(this).balance` however, for this circumstance, the `address(this).balance` is 1 ether, instead of 0. u may consider the following POC:

```
function testFail_deposit() public {
        manager.createVault(0.1 ether, 1 ether, 0, block.timestamp + 1 days, 0, false);
        address vault = manager.getVaults(0, 1)[0];
        StakingVault(vault).deposit{value: 1 ether}(address(0));
    }
```

Recommendation

comcat : Consider the below fix:

```
require(getAllEtherValue() <= round.maxVolume, "exceed");
```

The same fix can be applied to the `depositFor` function.

Client Response

Fixed

SSV-16: `StakingVault::exit` Implementation not completed

Category	Severity	Code Reference	Status	Contributor
Logical	Critical	• code/contracts/StakingVault.sol#L667-L687	Fixed	thereksfour, alansh, w2ning, comcat, jayphbee, Kong7ych3

Code

```
667:     function exit() external {
668:         require(terminated, "not terminated");
669:
670:         collectShares(msg.sender);
671:
672:         uint256 totalWithdraw;
673:         Vault.UserReceipt memory userReceipt = userReceipts[msg.sender];
674:         if (userReceipt.depositRound == latestRoundID) {
675:             totalWithdraw = userReceipt.pendingAmount;
676:         }
677:
678:         if (balanceOf(msg.sender) > 0) {
679:             totalWithdraw = totalWithdraw.add(
680:                 VaultMath.sharesToAsset(
681:                     balanceOf(msg.sender),
682:                     roundPricePerShare[latestRoundID]
683:                 )
684:             );
685:         }
686:     }
687:}
```

Description

thereksfour : When terminate() is called, the user can call exit() to exit the vault. However, in exit(), only the amount of the user's refund is calculated, and no refund is sent to the user This results in the user not being able to withdraw the

deposited ETH after the terminate

thereksfour : In exit(), the user's shares are converted to assets,

```
    function exit() external {
        require(terminated, "not terminated");

        collectShares(msg.sender);

        uint256 totalWithdraw;
        Vault.UserReceipt memory userReceipt = userReceipts[msg.sender];
        if (userReceipt.depositRound == latestRoundID) {
            totalWithdraw = userReceipt.pendingAmount;
        }

        if (balanceOf(msg.sender) > 0) {
            totalWithdraw = totalWithdraw.add(
                VaultMath.sharesToAsset(
                    balanceOf(msg.sender),
                    roundPricePerShare[latestRoundID]
                )
            );
        }
    }
```

but since the user sends shares to the contract in initiateWithdraw, these shares are not taken into account in the exit function, making the user lose these shares when calling exit().

```
    function initiateWithdraw(uint256 _shares)
        external
        nonReentrant
        notTerminated
    {
        ...
        _transfer(msg.sender, address(this), _shares);
```

alansh : The `exit` function is supposed to burn shares, transfer eth back to user, and update userReceipts.But somehow such logic is missing.

w2ning : Attackers can use multiple addresses to reuse the same Staking Token to obtain excess totalWithdraw amount. The impact is that the malicious contract may cause potential losses for normal users.

comcat : by design, when something emergency happens, the govern will terminate the contract by calling the `terminate` function. after that, the user will call the `exit` function to get their asset back. but when user call the `exit` function, they can not get their asset back, since it only collect shares for the user, and do nothing. and since the vault share is an ERC20 token, it can be transferred. so the user can actually call exit multiple times by transferring the share

to a new address.

jayphbee : There are a few problems of `exit` function implementation.

1. user's funds not transfered out. `totalWithdraw` is calculated, if it's greater than 0, funds should be transfered to user.
2. use wrong round price per share

```
                VaultMath.sharesToAsset(
                    balanceOf(msg.sender),
                    roundPricePerShare[latestRoundID]
                )
```

Here use the latest round price per share directly, should use `userReceipt.withdrawRound` instead.

3. user's share doesn't burn accordingly.

The impact is that if `exit` function not implement correctly user's funds can't be withrawn(due to 1), withdraw more/less than expected(due to 2) and can call `exit` more times to withdraw more funds(due to 3).

Kong7ych3 : In the StakingVault contract, the Gov role can terminate the operation of the agreement through the terminate function, which will set the terminated parameter in the contract to the true state, and can no longer reset the false state. When terminated is true, the user can no longer deposit and withdraw, and can only exit the agreement by calling the exit function. However, there is no refund logic in the exit function, and users can only receive share tokens. Assets such as CurveLP and native tokens in the protocol will be locked and cannot be withdrawn forever.

Recommendation

thereksfour :

```
    function exit() external {
        require(terminated, "not terminated");

        collectShares(msg.sender);

        uint256 totalWithdraw;
        Vault.UserReceipt memory userReceipt = userReceipts[msg.sender];
        if (userReceipt.depositRound == latestRoundID) {
            totalWithdraw = userReceipt.pendingAmount;
        }

        if (balanceOf(msg.sender) > 0) {
            totalWithdraw = totalWithdraw.add(
                VaultMath.sharesToAsset(
                    balanceOf(msg.sender),
                    roundPricePerShare[latestRoundID]
                )
            );
+           TransferHelper.safeTransferETH(msg.sender, totalWithdraw);
        }
    }
```

thereksfour : Consider using the logic of the completeWithdraw function in exit() to refund the user for the shares previously deposited

alansh : implement all the logic

w2ning : Transfer or burn token from msg.sender

Consider below fix in the `StakingVault.exit()` function

```
if (balanceOf(msg.sender) > 0) {
    totalWithdraw = totalWithdraw.add(
        VaultMath.sharesToAsset(
            balanceOf(msg.sender),
            roundPricePerShare[latestRoundID]
        )
    );

   // Transfer token from msg.sender
   _transfer(msg.sender, address(this), balanceOf(msg.sender);

   // Or burn token from msg.sender
   _burn(address(msg.sender), balanceOf(msg.sender);
}
```

comcat : complete the `exit` logic, to make sure user can actually withdraw their asset back. and make sure to burn those share from msg.sender.
jayphbee : rest the `pendingAmount`, burn the shares and transfer funds to user.

```
        if (balanceOf(msg.sender) > 0) {
            totalWithdraw = totalWithdraw.add(
                VaultMath.sharesToAsset(
                    balanceOf(msg.sender),
                    roundPricePerShare[userReceipt.withdrawRound]
                )
            );
        }
        userReceipts[msg.sender].pendingAmount = 0;
        _burn(msg.sender, balanceOf(msg.sender));
        if (totalWithdraw > 0) {
            TransferHelper.safeTransferETH(msg.sender, totalWithdraw);
        }
```

Kong7ych3 : It is recommended to add a refund function in the exit function.

Client Response

Fixed

SSV-17: `StakingVault::exit` always reverts

Category	Severity	Code Reference	Status	Contributor
Logical	Critical	• code/contracts/StakingVault.sol#L52-L55 • code/contracts/StakingVault.sol#L331-L332 • code/contracts/StakingVault.sol#L667-L670	Fixed	thereksfour, jayphbee, Kong7ych3

Code

```
52:    modifier notTerminated() {
53:        require(!terminated, "terminated");
54:        _;
55:    }

331:    function collectShares(address _user) public notTerminated {
332:        Vault.UserReceipt memory userReceipt = userReceipts[_user];

667:    function exit() external {
668:        require(terminated, "not terminated");
669:
670:        collectShares(msg.sender);
```

Description

thereksfour : exit() will call collectShares(), collectShares() has the notTerminated modifier. exit() requires terminated == true, but notTerminated requires terminated == false, so this causes exit() to not work

```
    function exit() external {
        require(terminated, "not terminated");

        collectShares(msg.sender);
    ...
    function collectShares(address _user) public notTerminated {
    ...
    modifier notTerminated() {
        require(!terminated, "terminated");
        _;
    }
```

This prevents the user from exiting the vault via exit(), thus leaving the user's assets locked in the contract

jayphbee : The `collectShare` function has `notTerminated` modifier.

```
function collectShares(address _user) public notTerminated
```

The `exit` function implemented like this:

```
    function exit() external {
        require(terminated, "not terminated");

        collectShares(msg.sender);
```

When `terminate` function is called, `terminate` variable is set to true. So the `collectShares` function will revert thus `exit` reverts.

The impact is that users can't withdraw their remaining balances after the staking vault is terminated due to `exit` always reverts.

Kong7ych3 : In the StakingVault contract, when the terminated state is true, it means that the protocol has been suspended, and the user can exit the protocol through the exit function. When the user calls the exit function, the collectShares operation will be performed first, and the collectShares function has a notTerminated modifier, which requires the terminated state to be false before the collectShares operation can be performed, but when the user performs the exit operation, the terminated state of the protocol is true. This will cause a conflict between the exit function and the collectShares function in the terminated state, making the exit function unavailable.

Recommendation

thereksfour : Consider implementing an internal _collectShares function

```
-    function collectShares(address _user) public notTerminated {
+    function _collectShares(address _user) internal {
         Vault.UserReceipt memory userReceipt = userReceipts[_user];

+    function collectShares(address _user) public notTerminated {
+        _collectShares(_user);
+    }
     function exit() external {
         require(terminated, "not terminated");

-        collectShares(msg.sender);
+        _collectShares(msg.sender);
```

jayphbee : I would suggest remove the `notTerminated` modifier for `collectShares` function.
Kong7ych3 : It is recommended to modify the collectShares function to internal visibility, and the exit function calls this function internally.

Client Response

Fixed

SSV-18: `StakingVault::getEtherOnLPMining` Call nested in the staticcall can not modify state

Category	Severity	Code Reference	Status	Contributor
Logical	Medium	• code/contracts/StakingVault.sol#L606-L627	Fixed	comcat

Code

```
606:     function getEtherOnLPMining() public view returns (uint256 etherOut) {
607:         if (lpMiningEnabled) {
608:             uint256 ldoBalance = ICurveGauge(CURVE_GAUGE).claimable_reward(
609:                 address(this),
610:                 LDO_TOKEN
611:             );
612:             ldoBalance = ERC20(LDO_TOKEN).balanceOf(address(this)).add(
613:                 ldoBalance
614:             );
615:             if (ldoBalance > 0) {
616:                 etherOut = etherOut.add(
617:                     IQuoter(QUOTER).quoteExactInputSingle(
618:                         LDO_TOKEN,
619:                         WETH,
620:                         3000,
621:                         ldoBalance,
622:                         0
623:                     )
624:                 );
625:             }
626:         }
627:     }
```

Description

comcat : in the StakingVault contract, there is a getter function:

```
function getEtherOnLPMining() public view returns (uint256 etherOut) {
        if (lpMiningEnabled) {
            uint256 ldoBalance =
                ICurveGauge(CURVE_GAUGE).claimable_reward(address(this), LDO_TOKEN);
           ...
            if (ldoBalance > 0) {
                etherOut = etherOut.add(
                    IQuoter(QUOTER).quoteExactInputSingle(
                        LDO_TOKEN, WETH, 3000, ldoBalance, 0
                    )
                );
            }
        }
    }
```

it is a view function, which means that all the external call inside this function should not modify the state, otherwise it will revert. however, when i check the CURVE_GAUGE and QUOTER, i just discover that:

```
@external
@nonreentrant('lock')
def claimable_reward(_addr: address, _token: address) -> uint256

function quoteExactInputSingle(
        address tokenIn,
        address tokenOut,
        uint24 fee,
        uint256 amountIn,
        uint160 sqrtPriceLimitX96
    ) public override returns (uint256 amountOut)
```

all of the above are not view function, but it will modify state. for the `claimable_reward`, it actually call the `claim_reward` first inside its function, which will transfer the reward to user and it modify the global state. for the `quoteExactInputSingle`, it actually use the try ... catch method, to run a swap function, which will modify state too.

Recommendation

comcat : change the related view function into non-view function.

```
function getEtherOnLPMining() public returns (uint256 etherOut)
function getEtherOnCurve() public returns (uint256 etherOut)
function getAllEtherInvested() public returns (uint256)
function getAllEtherValue() public returns (uint256)
```

Client Response

Fixed

SSV-19: `StakingVault::getEtherOnLPMining` calculation optimization for `etherOut`

Category	Severity	Code Reference	Status	Contributor
Gas Optimization	Informational	• code/contracts/StakingVault.sol#L616	Fixed	alansh

Code

```
616:                etherOut = etherOut.add(
```

Description

alansh : `etherOut` is not assigned before , so just doing assignment is enough.

Recommendation

alansh : Consider below fix in the `StakingVault.getEtherOnLPMining()` function

```
etherOut =   IQuoter(QUOTER).quoteExactInputSingle(
                        LDO_TOKEN,
                        WETH,
                        3000,
                        ldoBalance,
                        0
                    );
```

Client Response

Fixed

SSV-20: `StakingVault::initialize` should check whether `endTime` is as expected

Category	Severity	Code Reference	Status	Contributor
Logical	Low	• code/contracts/StakingVault.sol#L101	Fixed	iczc, comcat, jayphbee, Kong7ych3

Code

```
101:         require(roundInfo[0].endTime == 0, "Initialized");
```

Description

iczc : StakingVault uses roundInfo[0].endTime being 0 to determine if the contract is uninitialized, but the user can pass an endTime value of 0 during contract initialization, in which case the contract can be re-initialized.
comcat : in the stakingVault contract, the `initialize` function only checks the `require(roundInfo[0].endTime == 0, "Initialized");` as the only requirement to avoid re-initialize. however, this params is passed in the `createVault` function, which can only be called by the whitelisted publisher. which means that, if the publisher pass the `presaleEndTime` to 0, which leaves the stakingVault vulnerable to be re-initialized. u may consider the following:

```
function test_createVault() public {
        // publisher accident set the presaleEndTime=0
        manager.createVault(0.1 ether, 100 ether, 0, 0, 0, true);
        address vault = manager.getVaults(0, 1)[0];
        // the vault created by publisher is vulnerable to everyone to call the initialize again
        StakingVault(vault).initialize(
            0.1 ether,
            1000 ether,
            0,
            block.timestamp,
            address(this),
            stableSwap,
            curveLpToken,
            ldoRouter,
            gauge
        );
    }
```

recommendation change the requirement in the initialize:

```
    function initialize(
        uint256 _APY,
        uint256 _maxVolume,
        uint256 _minDeposit,
        uint256 _endTime,
        address _trader,
        address _stableSwap,
        address _curveLPToken,
        address _ldoRouter,
        address _gauge
    ) external {
        require(roundInfo[0].startTime == 0, "Initialized");
        ...
    }
```

jayphbee : There's no sanity check for `_presaleEndTime` and it's directly used to call the `StakingVault.initialize` function. If it's value is unexpectedly set to 0, `StakingVault.initialize` can be called once again due to:

```
        require(roundInfo[0].endTime == 0, "Initialized");

        roundInfo[0] = Vault.RoundInfo({
            APY: _APY,
            maxVolume: _maxVolume,
            filledVolume: 0,
            minDeposit: _minDeposit,
            startTime: block.timestamp,
            endTime: _endTime
        });
```

The address parameters `_stableSwap`, `_curveLPToken`, `_ldoRouter` and `_gauge` in the `initialize` function can be replaced by attacker's malicious address. When the protocol is interacting with one of the addresses, protocol's funds will lose.

Kong7ych3 : In the VaultManager contract, whitelist users can create a StakingVault contract through the createVault function, but it does not check whether the incoming `_presaleEndTime` is greater than the current time. If `_presaleEndTime` is mistakenly passed as 0, this will make `roundInfo[0].endTime` in the StakingVault contract be 0, resulting in the risk that StakingVault can be initialized again.

Recommendation

iczc : Verify the `_endTime` arg cannot be zero.

```
require(_endTime != 0, "Invalid");
```

comcat : n: in the stakingVault contract, the `initialize` function only checks the `require(roundInfo[0].endTime == 0, "Initialized");` as the only requirement to avoid re-initialize. however, this params is passed in the `createVault` function, which can only be called by the whitelisted publisher. which means that, if the publisher pass the `presaleEndTime` to 0, which leaves the stakingVault vulnerable to be re-initialized. u may consider the following:

```
function test_createVault() public {
        // publisher accident set the presaleEndTime=0
        manager.createVault(0.1 ether, 100 ether, 0, 0, 0, true);
        address vault = manager.getVaults(0, 1)[0];
        // the vault created by publisher is vulnerable to everyone to call the initialize again
        StakingVault(vault).initialize(
            0.1 ether,
            1000 ether,
            0,
            block.timestamp,
            address(this),
            stableSwap,
            curveLpToken,
            ldoRouter,
            gauge
        );
    }
```

recommendation change the requirement in the initialize:

```
    function initialize(
        uint256 _APY,
        uint256 _maxVolume,
        uint256 _minDeposit,
        uint256 _endTime,
        address _trader,
        address _stableSwap,
        address _curveLPToken,
        address _ldoRouter,
        address _gauge
    ) external {
        require(roundInfo[0].startTime == 0, "Initialized");
        ...
   }
```

Kong7ych3 : It is recommended to check whether `_presaleEndTime` is greater than the current time when creating a StakingVault.

Client Response

Fixed

SSV-21: `StakingVault::initiateWithdraw` should increase `withdrawingSharesInRound`

Category	Severity	Code Reference	Status	Contributor
Logical	Critical	• code/contracts/StakingVault.sol#L218-L260 • code/contracts/StakingVault.sol#L514-L528	Fixed	thereksfour

Code

```
218:     function initiateWithdraw(uint256 _shares)
219:         external
220:         nonReentrant
221:         notTerminated
222:     {
223:         require(_shares > 0, "less than zero");
224:
225:         Vault.UserReceipt storage receipt = userReceipts[msg.sender];
226:
227:         // Collect all shares
228:         if (receipt.pendingAmount > 0 || receipt.unredeemedShares > 0) {
229:             collectShares(msg.sender);
230:         }
231:
232:         require(balanceOf(msg.sender) >= _shares, "exceed");
233:
234:         if (receipt.withdrawRound == latestRoundID) {
235:             receipt.withdrawShares = receipt.withdrawShares.add(_shares);
236:             totalWithdrawingShares = totalWithdrawingShares.add(_shares);
237:             withdrawingSharesInRound = withdrawingSharesInRound.add(_shares);
238:         } else if (receipt.withdrawRound == 0) {
239:             receipt.withdrawShares = _shares;
240:             receipt.withdrawRound = latestRoundID;
241:             totalWithdrawingShares = totalWithdrawingShares.add(_shares);
242:             withdrawingSharesInRound = withdrawingSharesInRound.add(_shares);
243:         } else {
244:             // Withdraw previous round share first
245:             uint256 withdrawAmount = VaultMath.sharesToAsset(
246:                 receipt.withdrawShares,
247:                 roundPricePerShare[receipt.withdrawRound]
248:             );
249:
250:             _burn(address(this), receipt.withdrawShares);
251:
252:             totalWithdrawingShares = totalWithdrawingShares
253:                 .sub(receipt.withdrawShares)
254:                 .add(_shares);
255:             receipt.withdrawShares = _shares;
256:             receipt.withdrawableAmount = receipt.withdrawableAmount.add(
257:                 withdrawAmount
258:             );
259:             receipt.withdrawRound = latestRoundID;
```

```
260:         }

514:     function currentSharePrice() public view returns (uint256) {
515:         if (totalSupply() == 0) {
516:             return MULTIPLIER;
517:         }
518:
519:         (, , uint256 totalFee) = getFees();
520:
521:         uint256 etherAmount = getAllEtherValue()
522:             .sub(totalPendingAmount)
523:             .sub(withdrawableAmountInRound)
524:             .sub(totalFee);
525:         uint256 shareAmount = totalSupply().sub(withdrawingSharesInRound);
526:
527:         return etherAmount.mul(MULTIPLIER).div(shareAmount);
528:     }
```

Description

thereksfour : In initiateWithdraw, when the user deposits shares, withdrawingSharesInRound is increased, where withdrawingSharesInRound is used in the currentSharePrice() to calculate roundPricePerShare.

```
    function initiateWithdraw(uint256 _shares)
        external
        nonReentrant
        notTerminated
    {
        require(_shares > 0, "less than zero");

        Vault.UserReceipt storage receipt = userReceipts[msg.sender];

        // Collect all shares
        if (receipt.pendingAmount > 0 || receipt.unredeemedShares > 0) {
            collectShares(msg.sender);
        }

        require(balanceOf(msg.sender) >= _shares, "exceed");

        if (receipt.withdrawRound == latestRoundID) {
            receipt.withdrawShares = receipt.withdrawShares.add(_shares);
            totalWithdrawingShares = totalWithdrawingShares.add(_shares);
            withdrawingSharesInRound = withdrawingSharesInRound.add(_shares);
        } else if (receipt.withdrawRound == 0) {
            receipt.withdrawShares = _shares;
            receipt.withdrawRound = latestRoundID;
            totalWithdrawingShares = totalWithdrawingShares.add(_shares);
            withdrawingSharesInRound = withdrawingSharesInRound.add(_shares);
```

However, in initiateWithdraw, when withdrawRound < latestRoundID && withdrawRound ! = 0, there is no increase in withdrawingSharesInRound

```
        } else {
            // Withdraw previous round share first
            uint256 withdrawAmount = VaultMath.sharesToAsset(
                receipt.withdrawShares,
                roundPricePerShare[receipt.withdrawRound]
            );

            _burn(address(this), receipt.withdrawShares);

            totalWithdrawingShares = totalWithdrawingShares
                .sub(receipt.withdrawShares)
                .add(_shares);
            receipt.withdrawShares = _shares;
            receipt.withdrawableAmount = receipt.withdrawableAmount.add(
                withdrawAmount
            );
            receipt.withdrawRound = latestRoundID;
        }
```

Since withdrawingSharesInRound is small, shareAmount will be large in currentSharePrice(), resulting in a small result in currentSharePrice(), and thus roundPricePerShare will be small.

```
    function currentSharePrice() public view returns (uint256) {
        if (totalSupply() == 0) {
            return MULTIPLIER;
        }

        (, , uint256 totalFee) = getFees();

        uint256 etherAmount = getAllEtherValue()
            .sub(totalPendingAmount)
            .sub(withdrawableAmountInRound)
            .sub(totalFee);
        uint256 shareAmount = totalSupply().sub(withdrawingSharesInRound);

        return etherAmount.mul(MULTIPLIER).div(shareAmount);
    }
```

Recommendation

thereksfour : Change to

```
        } else {
            // Withdraw previous round share first
            uint256 withdrawAmount = VaultMath.sharesToAsset(
                receipt.withdrawShares,
                roundPricePerShare[receipt.withdrawRound]
            );

            _burn(address(this), receipt.withdrawShares);
+           withdrawingSharesInRound = withdrawingSharesInRound.add(_shares);
            totalWithdrawingShares = totalWithdrawingShares
                .sub(receipt.withdrawShares)
                .add(_shares);
            receipt.withdrawShares = _shares;
            receipt.withdrawableAmount = receipt.withdrawableAmount.add(
                withdrawAmount
            );
            receipt.withdrawRound = latestRoundID;
        }
```

Client Response

Fixed

SSV-22: `StakingVault::sellAllLDO` Incorrect recipient address

Category	Severity	Code Reference	Status	Contributor
Logical	Critical	• code/contracts/StakingVault.sol#L500	Fixed	jayphbee, Kong7ych3

Code

```
500:                        recipient: msg.sender,
```

Description

jayphbee : The `recipient` address in the `ExactInputSingleParams` struct should be `address(this)` instead of `msg.sender`.

```
                ISwapRouter.ExactInputSingleParams memory params = ISwapRouter
                    .ExactInputSingleParams({
                        tokenIn: LDO_TOKEN,
                        tokenOut: WETH,
                        fee: 3000,
                        recipient: msg.sender,
                        deadline: block.timestamp,
                        amountIn: balance,
                        amountOutMinimum: minReceived,
                        sqrtPriceLimitX96: 0
                    });

                ISwapRouter(LDO_ROUTER).exactInputSingle(params);
```

The impact is that the funds received when selling LDO goes to `msg.sender` instead the `StakingVault` contract, which will lead to protocol insolvent.

Kong7ych3 : In the StakingVault contract, the sellAllLDO function is used to sell LDO tokens into ETH through Uniswap, but the recipient address of the swap parameter is incorrectly filled in as `msg.sender`, which will cause the settlement and termination operations to fail.

Recommendation

jayphbee : Use `address(this)` as the recipient address.

Kong7ych3 : It is recommended to change the recipient address from `msg.sender` to `address(this)`.

Client Response

Fixed

SSV-23: `StakingVault::sellAllLDO` `WETH` is not unwrapped

Category	Severity	Code Reference	Status	Contributor
Logical	Critical	• code/contracts/StakingVault.sol#L498	Fixed	jayphbee

Code

```
498:                     tokenOut: WETH,
```

Description

jayphbee : In the `sellAllLDO` function, all the LDO will be sold and receive the WETH token.

```
                ISwapRouter.ExactInputSingleParams memory params = ISwapRouter
                    .ExactInputSingleParams({
                        tokenIn: LDO_TOKEN,
                        tokenOut: WETH,
                        fee: 3000,
                        recipient: msg.sender,
                        deadline: block.timestamp,
                        amountIn: balance,
                        amountOutMinimum: minReceived,
                        sqrtPriceLimitX96: 0
                    });

                ISwapRouter(LDO_ROUTER).exactInputSingle(params);
```

But the WETH token isn't unwrapped in the `sellAllLDO` function or elsewhere. The impact is that WETH token is stucked in `StakingVault` contract and can't be withdrawn.

Recommendation

jayphbee : Unwrap WETH token using `WETH.withdraw` after the swap is finished.

Client Response

Fixed

SSV-24: `StakingVault::settlement` Anyone can provide `_minLPMint` and `_minEtherReceived` leading to sandwich attack

Category	Severity	Code Reference	Status	Contributor
Race Condition	Medium	• code/contracts/StakingVault.sol#L358 • code/contracts/StakingVault.sol#L406-L414 • code/contracts/StakingVault.sol#L432-L435	Fixed	thereksfour, jayphbee, Kong7ych3

Code

```
358:    function settlement(uint256 _minLPMint, uint256 _minEtherReceived)

406:        if (assets.etherNeedToInvest > 0) {
407:            uint256 lpMinted = IStableSwap(STABLE_SWAP).add_liquidity{
408:                value: assets.etherNeedToInvest
409:            }([assets.etherNeedToInvest, 0], _minLPMint);
410:
411:            TransferHelper.safeApprove(CURVE_LP_TOKEN, CURVE_GAUGE, lpMinted);
412:
413:            ICurveGauge(CURVE_GAUGE).deposit(lpMinted);
414:        }

432:            require(
433:                _minEtherReceived.add(etherBefore) >= address(this).balance,
434:                "slippage"
435:            );
```

Description

thereksfour : The settlement can be called by anyone, and tokens are exchanged in STABLE_SWAP as needed in the settlement, where the parameter _minLPMint/_minEtherReceived for slippage control is provided by the caller.

```
    function settlement(uint256 _minLPMint, uint256 _minEtherReceived)
        public
        notTerminated
    {
```

When etherNeedToSell > soldByLDO, the result of calc_token_amount is used for slippage control.

```
        if (assets.etherNeedToSell > assets.soldByLDO) {
            uint256 etherBefore = address(this).balance;
            assets.etherNeedToSell = assets.etherNeedToSell.sub(
                assets.soldByLDO
            );
            uint256 lpNeeded = IStableSwap(STABLE_SWAP).calc_token_amount(
                [assets.etherNeedToSell, 0],
                false
            );

            ICurveGauge(CURVE_GAUGE).withdraw(lpNeeded);
            IStableSwap(STABLE_SWAP).remove_liquidity_imbalance(
                [assets., 0],
                lpNeeded
            );

            require(
                _minEtherReceived.add(etherBefore) >= address(this).balance,
                "slippage"
            );
        }
```

When etherNeedToInvest > 0, the slippage control is entirely determined by the _minLPMint parameter, which can be used by malicious callers to perform sandwich attacks.

```
        if (assets.etherNeedToInvest > 0) {
            uint256 lpMinted = IStableSwap(STABLE_SWAP).add_liquidity{
                value: assets.etherNeedToInvest
            }([assets.etherNeedToInvest, 0], _minLPMint);

            TransferHelper.safeApprove(CURVE_LP_TOKEN, CURVE_GAUGE, lpMinted);

            ICurveGauge(CURVE_GAUGE).deposit(lpMinted);
        }
```

jayphbee : After reach the `round.endTime`, anyone can call `settlement`.

```
  require(block.timestamp >= round.endTime, "too early");
```

There are `_minLPMint` and `_mintEtherReceived` parameters to control the slippage when the protocol interacting to curve pool. Unfortunately, MEV reseacher can pass arbitrary value to them and then exploit them to build a sandwich

transaction to make profit.

The impact is that the `settlement` suffer un-controlled slippage when interacting to curve pool.

Kong7ych3 : In the settlement function of the StakingVault contract, when the protocol removes liquidity from the Curve Pool, a slippage check will be performed to ensure that the pool has not been manipulated. But unfortunately the slippage check depends on the `_minEtherReceived` parameter passed in by the user, and any user can call the settlement function to pass in any `_minEtherReceived` value. An attacker can pass `_minEtherReceived` as 0 to manipulate the pool at will and bypass the slippage check.

Recommendation

thereksfour : Use oracle to get price and set minimum out or restrict only owner/governance to provide this parameter and call this function.

jayphbee : I would suggest only the trusted third party can call `settlement` function.

Kong7ych3 : It is recommended to calculate the amount of received ETH tokens through the stable LP price.

There are two ways to obtain the stable LP price, one is obtained through the get_virtual_price function of Curve Pool, and the other is obtained through the LP price calculation algorithm released by the Conic team (please refer to ref[1]).

But it should be noted that the get_virtual_price function in the ETH/stETH pool is affected by the reentrancy vulnerability, you can refer to the solution in ref[2] for implementation. The algorithm announced by the Conic team has not been widely verified, and the gas efficiency seems to be too low.

The following takes get_virtual_price as an example to check for slippage:

```
etherOut = (lpNeeded * IStableSwap(STABLE_SWAP).get_virtual_price()) / 1e36
(etherOut + etherBefore) * (1 - slippage) <= address(this).balance <= (etherOut + etherBefore) * (1
+ slippage)
```

ref: [1] https://conic.finance/whitepaper.pdf Chapters 3.1 and 3.2 [2] https://chainsecurity.com/curve-lp-oracle-manipulation-post-mortem/

Client Response

Fixed by adding `onlyGovernance` modifier to the function.

SSV-25: `StakingVault::settlement` Incorrect handling of `etherNeedToSell`

Category	Severity	Code Reference	Status	Contributor
Logical	Critical	• code/contracts/StakingVault.sol#L416-L436	Fixed	thereksfour

Code

```
416:         if (assets.etherNeedToSell > assets.soldByLDO) {
417:             uint256 etherBefore = address(this).balance;
418:             assets.etherNeedToSell = assets.etherNeedToSell.sub(
419:                 assets.soldByLDO
420:             );
421:             uint256 lpNeeded = IStableSwap(STABLE_SWAP).calc_token_amount(
422:                 [assets.etherNeedToSell, 0],
423:                 false
424:             );
425:
426:             ICurveGauge(CURVE_GAUGE).withdraw(lpNeeded);
427:             IStableSwap(STABLE_SWAP).remove_liquidity_imbalance(
428:                 [assets.etherNeedToSell, 0],
429:                 lpNeeded
430:             );
431:
432:             require(
433:                 _minEtherReceived.add(etherBefore) >= address(this).balance,
434:                 "slippage"
435:             );
436:         }
```

Description

thereksfour : The process of handling etherNeedToSell in settlement is incorrect, Consider the following scenario totalWithdraw = 17 soldByLDO = 5 totalPendingAmount = 10 In the calculation below totalPendingAmount = 10 + 5 = 15 etherNeedToSell = 17 - 15 = 2 Since etherNeedToSell < soldByLDO, lp will not be sold, which is obviously incorrect, which will cause users to be unable to withdraw ETH due to insufficient balance

```
        assets.soldByLDO = sellAllLDO();
        totalPendingAmount = totalPendingAmount.add(assets.soldByLDO);
        ...
        if (assets.totalWithdraw > totalPendingAmount) {
            assets.etherNeedToSell = assets.totalWithdraw.sub(
                totalPendingAmount
            );
        }
       ...
        if (assets.etherNeedToSell > assets.soldByLDO) {
            uint256 etherBefore = address(this).balance;
            assets.etherNeedToSell = assets.etherNeedToSell.sub(
                assets.soldByLDO
            );
            uint256 lpNeeded = IStableSwap(STABLE_SWAP).calc_token_amount(
                [assets.etherNeedToSell, 0],
                false
            );

            ICurveGauge(CURVE_GAUGE).withdraw(lpNeeded);
            IStableSwap(STABLE_SWAP).remove_liquidity_imbalance(
                [assets.etherNeedToSell, 0],
                lpNeeded
            );

            require(
                _minEtherReceived.add(etherBefore) >= address(this).balance,
                "slippage"
            );
        }
```

Recommendation

thereksfour : Change to

```
        if (assets.etherNeedToSell > 0) {
            uint256 etherBefore = address(this).balance;
            uint256 lpNeeded = IStableSwap(STABLE_SWAP).calc_token_amount(
                [assets.etherNeedToSell, 0],
                false
            );

            ICurveGauge(CURVE_GAUGE).withdraw(lpNeeded);
            IStableSwap(STABLE_SWAP).remove_liquidity_imbalance(
                [assets.etherNeedToSell, 0],
                lpNeeded
            );

            require(
                _minEtherReceived.add(etherBefore) >= address(this).balance,
                "slippage"
            );
        }
```

Client Response

Fixed the issue with a new logic

SSV-26: `StakingVault::settlement` The number of shares minted is excessive

Category	Severity	Code Reference	Status	Contributor
Logical	Medium	• code/contracts/StakingVault.sol#L452-L457	Fixed	thereksfour

Code

```
452:         uint256 mintShares = VaultMath.assetToShares(
453:             totalPendingAmount,
454:             sharePrice
455:         );
456:
457:         _mint(address(this), mintShares);
```

Description

thereksfour : In settlement, shares are minted based on the amount of ETH deposited by the user in this round (totalPendingAmount) However, because totalPendingAmount is added to soldByLDO, the totalPendingAmount is inflated, resulting in too many shares being minted.

```
        totalPendingAmount = totalPendingAmount.add(assets.soldByLDO);
        ...
        uint256 mintShares = VaultMath.assetToShares(
            totalPendingAmount,
            sharePrice
        );

        _mint(address(this), mintShares);
```

The excess shares will not be claimed by the user, but will be locked in the contract. Since totalSupply is inflated, the price of each share is diluted in currentSharePrice

```
    function currentSharePrice() public view returns (uint256) {
        if (totalSupply() == 0) {
            return MULTIPLIER;
        }

        (, , uint256 totalFee) = getFees();

        uint256 etherAmount = getAllEtherValue()
            .sub(totalPendingAmount)
            .sub(withdrawableAmountInRound)
            .sub(totalFee);
        uint256 shareAmount = totalSupply().sub(withdrawingSharesInRound);

        return etherAmount.mul(MULTIPLIER).div(shareAmount);
    }
```

Recommendation

thereksfour :

```
        totalPendingAmount = totalPendingAmount.add(assets.soldByLDO);
        ...
        uint256 mintShares = VaultMath.assetToShares(
-           totalPendingAmount,
+          totalPendingAmount.sub(assets.soldByLDO),
            sharePrice
        );
```

Client Response

Fixed the issue with a new logic

SSV-27: `StakingVault::settlement` does not send premium amount ether to `OptionsTrading`

Category	Severity	Code Reference	Status	Contributor
Logical	Low	• code/contracts/StakingVault.sol#L460	Fixed	jayphbee

Code

```
460:        uint256 amount = OptionsTrading(optionsTrading).roll(
```

Description

jayphbee : `OptionsTrading.roll` is designed payable to receive ether from `StakingVault` contract when calling `settlement` function.

```
        uint256 amount = OptionsTrading(optionsTrading).roll(
            latestRoundID,
            assets.premium
        );
```

But when calling the `OptionsTrading.roll` in `settelment` function there's no value send to `OptionsTrading` contract.

Recommendation

jayphbee : Send premium amount of ether to `OptionsTrading` contract in the `OptionsTrading.roll` function.

```
        uint256 amount = OptionsTrading(optionsTrading).roll{value: assets.premium}(
            latestRoundID,
            assets.premium
        );
```

Client Response

Fixed the issue with a new logic

SSV-28: `StakingVault::settlement` wrong parameter order when calling `VaultMath.getPremium`

Category	Severity	Code Reference	Status	Contributor
Code Style	Low	• code/contracts/libraries/VaultMath.sol#L71-L77 • code/contracts/StakingVault.sol#L384-L388	Fixed	alansh, Kong7ych3

Code

```
71:     function getPremium(
72:         uint256 _volume,
73:         uint256 _APY,
74:         uint256 _period
75:     ) internal pure returns (uint256) {
76:         return (_volume * _APY * _period) / PERCENTAGE / DAYS_IN_YEAR;
77:     }

384:         assets.premium = VaultMath.getPremium(
385:             params.cycle,
386:             assets.volumeInRound,
387:             round.APY
388:         );
```

Description

alansh : The parameter order doesn't match the signature order.
Kong7ych3 : In the settlement function of the StakingVault contract, it will calculate the premium through the getPremium function based on the values of cycle, volumeInRound and APY. However, when calling the getPremium function, an incorrect parameter is passed, as shown in the following code block. Although it does not affect the final calculation result, parameters should still be passed according to the specification.

```
assets.premium = VaultMath.getPremium(
    params.cycle,
    assets.volumeInRound,
    round.APY
);

function getPremium(
    uint256 _volume,
    uint256 _APY,
    uint256 _period
) internal pure returns (uint256) {
    return (_volume * _APY * _period) / PERCENTAGE / DAYS_IN_YEAR;
}
```

Recommendation

alansh : Consider below fix in the `StakingVault.settlement()` function

```
assets.premium = VaultMath.getPremium(
            assets.volumeInRound,
            round.APY,
            params.cycle
        );
```

Kong7ych3 : It is recommended to refer to the parameters accepted by the getPremium function, first pass in `assets.volumeInRound`, then pass in `round.APY`, and finally pass in `params.cycle`.

Client Response

Fixed

SSV-29: `StakingVault` Missing events for critical parameters

Category	Severity	Code Reference	Status	Contributor
Code Style	Informational	• code/contracts/StakingVault.sol#L629 • code/contracts/StakingVault.sol#L633 • code/contracts/StakingVault.sol#L637	Acknowledged	comcat

Code

```
629:    function setPriceFeeder(address _priceFeeder) external onlyGovernance {

633:    function setSlippage(uint256 _slippage) external onlyGovernance {

637:    function terminate(uint256 _minEtherReceived)
```

Description

comcat : inside the `stakingVault` contract, there are some onlyGovernance functions, but lack corresponding events

```
function setSlippage(uint256 _slippage) external onlyGovernance
function setPriceFeeder(address _priceFeeder) external onlyGovernance
function terminate(uint256 _minEtherReceived) external notTerminated onlyGovernance
```

Recommendation

comcat : add corresponding events for those onlyGovernance functions

```
function setPriceFeeder(address _priceFeeder) external onlyGovernance {
        priceFeeder = _priceFeeder;
        emit PriceFeederSet(_priceFeeder);
    }

    function setSlippage(uint256 _slippage) external onlyGovernance {
        slippage = _slippage;
        emit SlippageSet(_slippage);
    }

    function terminate(uint256 _minEtherReceived) external notTerminated onlyGovernance {
        ...
        terminated = true;
        emit Terminated();
    }
```

Client Response

Acknowledged

SSV-30: `StakingVault` Risk of price manipulation

Category	Severity	Code Reference	Status	Contributor
Price Manipulation	Critical	• code/contracts/StakingVault.sol#L407 • code/contracts/StakingVault.sol#L421 • code/contracts/StakingVault.sol#L599 • code/contracts/StakingVault.sol#L617	Fixed	Kong7ych3

Code

```
407:            uint256 lpMinted = IStableSwap(STABLE_SWAP).add_liquidity{

421:            uint256 lpNeeded = IStableSwap(STABLE_SWAP).calc_token_amount(

599:            etherOut = IStableSwap(STABLE_SWAP).calc_withdraw_one_coin(

617:                    IQuoter(QUOTER).quoteExactInputSingle(
```

Description

Kong7ych3 : In the StakingVault contract, the getEtherOnCurve function calculates the amount of ETH liquidity the protocol has in the Curve Pool through the calc_withdraw_one_coin function. The getEtherOnLPMining function calculates the amount of ETH tokens that can be exchanged for LDO tokens through the quoteExactInputSingle function.

However, it is well known that the calculations performed by the calc_withdraw_one_coin function and the quoteExactInputSingle function are easily affected by the real-time amount of tokens in the pool. Malicious users only need to perform large swap operations in Curve Pool and Uniswap Pool through flashloan to manipulate the results of calc_withdraw_one_coin and quoteExactInputSingle.

Once the getEtherOnCurve and getEtherOnLPMining functions are manipulated, it will directly affect the results of the getAllEtherInvested, getAllEtherValue and currentSharePrice functions.

And in the deposit function, check the maxVolume through the getAllEtherValue function; in the settlement function, use the currentSharePrice and getAllEtherInvested functions to calculate the share price and the filledVolume parameter value of the next round respectively. This will make the core functions of the protocol extremely vulnerable to manipulation. The attacker can manipulate the price at will through flashloan to attack.

The calculation of lpNeeded in the settlement function is also at risk of being manipulated.

Recommendation

Kong7ych3 : To ensure that the results obtained by getEtherOnLPMining and getEtherOnCurve are correct, you should first ensure that the Curve Pool and Uniswap Pool are not manipulated.

For the getEtherOnCurve function, we can use the stable LP price to calculate the amount of ETH assets owned by the Curve Pool, as follows: `etherOut = (lpBalance * stableLpPrice) / decimal` There are two ways to obtain the stable LP price, one is obtained through the get_virtual_price function of Curve Pool, and the other is obtained through the LP price calculation algorithm released by the Conic team (please refer to ref[1]).

But it should be noted that the get_virtual_price function in the ETH/stETH pool is affected by the reentrancy vulnerability, you can refer to the solution in ref[2] for implementation. The algorithm announced by the Conic team has not been widely verified, and the gas efficiency seems to be too low.

For the implementation of get_virtual_price, you can refer to the following code:

```
etherOut = (lpBalance * IStableSwap(STABLE_SWAP).get_virtual_price()) / 1e36
```

To ensure the correctness of getEtherOnLPMining, it is only necessary to compare the price of Chainlink's LDO/ETH oracle with the quantity returned by quoteExactInputSingle. If the difference between the two values is too large, it proves that the pool is likely to be manipulated.

ref: [1] https://conic.finance/whitepaper.pdf Chapters 3.1 and 3.2 [2] https://chainsecurity.com/curve-lp-oracle-manipulation-post-mortem/

The calculation of lpNeeded in the settlement function is also at risk of being manipulated. At this point we only need to pass `lpNeeded = etherNeedToSell / get_virtual_price`.

Client Response

Fixed

SSV-31: `StakingVault` does not work when `curveEnabled` is `false`

Category	Severity	Code Reference	Status	Contributor
Logical	Medium	• code/contracts/StakingVault.sol#L411-L413 • code/contracts/StakingVault.sol#L426 • code/contracts/StakingVault.sol#L645-L647	Fixed	alansh, comcat

Code

```
411:             TransferHelper.safeApprove(CURVE_LP_TOKEN, CURVE_GAUGE, lpMinted);
412:
413:             ICurveGauge(CURVE_GAUGE).deposit(lpMinted);

426:             ICurveGauge(CURVE_GAUGE).withdraw(lpNeeded);

645:         uint256 lpAmount = ICurveGauge(CURVE_GAUGE).balanceOf(address(this));
646:
647:         ICurveGauge(CURVE_GAUGE).withdraw(lpAmount);
```

Description

alansh : As `CURVE_GAUGE` is only set when `lpMiningEnabled` is `true`, but references to `CURVE_GAUGE` don't check this precondition in `settlement` and `terminate` function. L645 will always fail when `lpMiningEnabled` is false.

comcat : when create a `stakingVault`, the publisher can set it to non curveEnabled, by passing the param: `_curveEnabled = false`. Then, inside the stakingVault `initialize` method, it will judge the `lpMiningEnabled` to set the corresponding `CURVE_GAUGE, LDO_TOKEN, LDO_ROUTER` address. which means that, if the `_curveEnabled = false`, then the stakingVault's `CURVE_GAUGE, LDO_TOKEN, LDO_ROUTER` address are left to be `address zero`. however, inside the `settlement` function, it doesn't check whether `CURVE_GAUGE, LDO_TOKEN, LDO_ROUTER` address are zero or not, it has the branch to call `ICurveGauge(CURVE_GAUGE).deposit(lpMinted);` which will cause the settlement revert. that is:

```
constructor(
        ...
        bool _curveEnabled
    ) ERC20("Shield Vault Token", "Shield Vault Token") {
       ...
        if (_curveEnabled) {
            curveEnabled = _curveEnabled;
            lpMiningEnabled = true;
        }
        ...
    }
function initialize(
       ...
    ) external {
        ...
        if (lpMiningEnabled) {
            require(ICurveGauge(_gauge).lp_token() == CURVE_LP_TOKEN, "invalid gauge");
            CURVE_GAUGE = _gauge;
            LDO_TOKEN = ICurveGauge(_gauge).reward_tokens(0);
            LDO_ROUTER = _ldoRouter;
        }
    }
function settlement(uint256 _minLPMint, uint256 _minEtherReceived)
        public
        notTerminated
    {
        ...
        if (assets.etherNeedToInvest > 0) {
            ...
            ICurveGauge(CURVE_GAUGE).deposit(lpMinted);
        }

        if (assets.etherNeedToSell > assets.soldByLDO) {
            ...
            ICurveGauge(CURVE_GAUGE).withdraw(lpNeeded);
           ...
        }
    ...
    }
```

you may refer to the following poc:

```
function test_deposit() public {
        manager.createVault(0.1 ether, 1 ether, 0, block.timestamp, 0, false);
        address vault = manager.getVaults(0, 1)[0];
        StakingVault(vault).depositFor{value: 0.1 ether}(address(this));
        StakingVault(vault).settlement(0, 0);
}
```

Recommendation

alansh : Always check `lpMiningEnabled` before calling `CURVE_GAUGE`, like what `sellAllLDO` does.
comcat : since the lpMining can be turned off by design, it should check address `CURVE_GAUGE, LDO_TOKEN, LDO_ROUTER` address are zero or not inside the settlement function. if it is zero, it should just skip it.

```
function settlement(uint256 _minLPMint, uint256 _minEtherReceived)
        public
        notTerminated
    {
        ...
        if (assets.etherNeedToInvest > 0 && GURVE_GAUGE != address(0)) {
            ...
            ICurveGauge(CURVE_GAUGE).deposit(lpMinted);
        }

        if (assets.etherNeedToSell > assets.soldByLDO && CURVE_GAUGE != address(0)) {
            ...
            ICurveGauge(CURVE_GAUGE).withdraw(lpNeeded);
           ...
        }
    ...
    }
```

Client Response

Removed `curveEnabled`

SSV-32: `VaultManager` Gas optimization by using `immutable`

Category	Severity	Code Reference	Status	Contributor
Gas Optimization	Informational	• code/contracts/VaultManager.sol# L7-L16	Fixed	comcat

Code

```
7:    address private governance;
8:
9:    address private CURVE_LP_TOKEN;
10:    address private STABLE_SWAP;
11:    address private CURVE_GAUGE;
12:    address private UNISWAP_QUOTER;
13:    address private LDO_ROUTER;
14:
15:    address private WETH;
16:    address private STETH;
```

Description

comcat : in the `vaultManager` contract, some global params can be set as immutable to save gas. for example:

```
    address private governance;
    address private CURVE_LP_TOKEN;
    address private STABLE_SWAP;
    address private CURVE_GAUGE;
    address private UNISWAP_QUOTER;
    address private LDO_ROUTER;

    address private WETH;
    address private STETH;
```

Recommendation

comcat : since those params will not change, it is better to set it as immutable to save gas:

```
    address private immutable governance;

    address private immutable CURVE_LP_TOKEN;
    address private immutable STABLE_SWAP;
    address private immutable CURVE_GAUGE;
    address private immutable UNISWAP_QUOTER;
    address private immutable LDO_ROUTER;

    address private immutable WETH;
    address private immutable STETH;
```

Client Response

Fixed

SSV-33: `VaultManager` Missing events and caps for the `setManagementFeeRate` and `setPerformanceFeeRate`

Category	Severity	Code Reference	Status	Contributor
Code Style	Informational	• code/contracts/VaultManager.sol# L169-L175	Acknowledged	comcat, jayphbee

Code

```
169:     function setManagementFeeRate(uint256 _rate) external onlyGovernance {
170:         managementFeeRate = _rate;
171:     }
172:
173:     function setPerformanceFeeRate(uint256 _rate) external onlyGovernance {
174:         performanceFeeRate = _rate;
175:     }
```

Description

comcat : inside the VaultManager contract, the governor has the power to set the `management fee rate` and `performance fee rate`. and there is no restriction to limit the fee rate value, which means that it can be set as high as possible, while the publisher has no way to change it, but only accept it. also, these two function miss the corresponding events.
jayphbee : When `managementFeeRate` and `performanceFeeRate` updated, there's no corresponding event emitted. It is the best practice to emit an event when sysytem parameter changed.

Recommendation

comcat : set cap for the `management fee rate` and `performance fee rate`, as well, add the missing events

```
function setManagementFeeRate(uint256 _rate) external onlyGovernance {
        require(_rate < MANAGEMENT_FEE_RATE_CAP, "exceed the cap");
        managementFeeRate = _rate;
        emit SetManagementFeeRate(_rate);
    }

    function setPerformanceFeeRate(uint256 _rate) external onlyGovernance {
        require(_rate < PERFORMANCE_FEE_RATE_CAP, "exceed the cap");
        performanceFeeRate = _rate;
        emit SetPerformanceFeeRate(_rate);
    }
```

jayphbee : emit an event when `managementFeeRate` and `performanceFeeRate` updated.

Client Response

Acknowledged

SSV-34: `VaultStorage` Gas optimization by reorganizing storage layout

Category	Severity	Code Reference	Status	Contributor
Gas Optimization	Informational	• code/contracts/structs/VaultStorage.sol#L6	Fixed	comcat

Code

```
6:abstract contract VaultStorage {
```

Description

comcat : vault storage is a contract that contains lots of global params, which inherited by StakingVault. those global params could be re-org, so that to save some gas.

Recommendation

comcat : the change is:

1. add constant to the address QUOTER and address CURVE_ETH
2. move bool after address, so that they can be compacted into 1 slot

```
    uint256 internal constant MULTIPLIER = 1e18;

    address internal constant QUOTER = 0xb27308f9F90D607463bb33eA1BeBb41C27CE5AB6;

    address internal constant CURVE_ETH = 0xEeeeeEeeeEeEeeEeEeEeeEEEeeeeEeeeeeeeEEeE;

    address internal governance;

    address public publisher;

    address internal priceFeeder;

    address public optionsTrading;

    address public broker;

    address internal STABLE_SWAP;

    address internal CURVE_LP_TOKEN;

    address internal CURVE_GAUGE;

    address internal LDO_TOKEN;

    address internal LDO_ROUTER;

    bool public terminated;

    address internal STETH;

    bool internal curveEnabled;

    address internal WETH;

    bool internal lpMiningEnabled;

    uint256 public minDeposit;

    uint256 public latestRoundID;

    uint256 public totalPendingAmount;

    uint256 public totalWithdrawingShares;
```

```
uint256 public withdrawingSharesInRound;

    uint256 public totalWithdrawableAmount;

    uint256 public withdrawableAmountInRound;

    uint256 public managementFeeRate;

    uint256 public performanceFeeRate;

    uint256 internal slippage = 995000; // 99.5%
```

Client Response

Fixed

SSV-35: `receive()` is not implemented in `OptionTrading` and `StakingVault` contract

Category	Severity	Code Reference	Status	Contributor
Logical	Medium	• code/contracts/OptionsTrading.sol#L9 • code/contracts/structs/VaultStorage.sol#L24 • code/contracts/StakingVault.sol#L114 • code/contracts/OptionsTrading.sol#L115-L118 • code/contracts/StakingVault.sol#L464-L466 • code/contracts/StakingVault.sol#L465 • code/contracts/StakingVault.sol#L648	Fixed	comcat, jayphbee

Code

```
9:contract OptionsTrading is ReentrancyGuard {

24:    address public optionsTrading;

114:        optionsTrading = address(trading);

115:            TransferHelper.safeTransferETH(
116:                vault,
117:                address(this).balance.sub(_premium)
118:            );

464:        if (amount > 0) {
465:            TransferHelper.safeTransferETH(optionsTrading, amount);
466:        }

465:            TransferHelper.safeTransferETH(optionsTrading, amount);

648:        IStableSwap(STABLE_SWAP).remove_liquidity_imbalance(
```

Description

comcat : By design, the OptionTrading contract can receive ether, as it states in StakingVault's `settlement` function:

```
function settlement(uint256 _minLPMint, uint256 _minEtherReceived)
        public
        notTerminated
    {
       ...
        uint256 amount =
            OptionsTrading(optionsTrading).roll(latestRoundID, assets.premium);
        if (amount > 0) {
            TransferHelper.safeTransferETH(optionsTrading, amount);
        }
    }
```

However, when check the contract OptionTrading, it doesn't have any receive function or fallback payable function. so, basically `TransferHelper.safeTransferETH(optionsTrading, amount);` will revert, as no receive function exists.

comcat : In the `terminate` function, it will remove lp from curve to get Ether back. however, there is no receive function in the stakingVault, which will cause the `remove_liquidity_imbalance` function revert. the terminate funciton:

```
function terminate(uint256 _minEtherReceived) external notTerminated onlyGovernance {
        ...
         ICurveGauge(CURVE_GAUGE).withdraw(lpAmount);
         IStableSwap(STABLE_SWAP).remove_liquidity_imbalance(
             [_minEtherReceived.sub(sold), 0], lpAmount
         ); /// will trigger Ether transfer, as no receive function exists, it will revert
        ...
    }
```

you may refer to the following POC:

```
function test_deposit() public {
        manager.createVault(0.1 ether, 1 ether, 0, block.timestamp, 0, true);
        address vault = manager.getVaults(0, 1)[0];
        StakingVault(vault).depositFor{value: 0.1 ether}(address(this));
        StakingVault(vault).settlement(0, 0);
        // (uint256 share,) = StakingVault(vault).accountShare(address(this));
        // StakingVault(vault).initiateWithdraw(share);
        StakingVault(vault).terminate(0.01 ether);
        StakingVault(vault).exit();
}
```

jayphbee : In the `OptionsTrading.roll` function, ethers may be sent to the StakingVault contract.

```
        if (address(this).balance > _premium) {
            TransferHelper.safeTransferETH(
                vault,
                address(this).balance.sub(_premium)
            );
         }
```

In the `StakingVault.settlement` function, ethers may be sent to the `OptionTarding` contract.

```
         uint256 amount = OptionsTrading(optionsTrading).roll(
             latestRoundID,
             assets.premium
         );
         if (amount > 0) {
             TransferHelper.safeTransferETH(optionsTrading, amount);
         }
```

But both `OptionsTrading` and `StakingVault` doesn't implement `receive()` thus ethers transfer will be reverted between them. Further more, `IStableSwap.remove_liquidity_imbalance` called in `sellement` and `terminate` function will send ether to `StakingVault` contract. This will lead to revert when calling `sellement` and `terminate` function.

The impact is that there will be unexpectedly revert when calling `StakingVault.settlement` and `StakingVault.terminate`.

Recommendation

comcat : Add receive function for contract OptionTrading

```
receive() external payable {
        require(msg.sender == vault, "not allowed");
    }
```

change vaultStorage :

```
// address public optionsTrading;
address payable public optionsTrading;
```

change StakingVault:

```
function initialize(
        uint256 _APY,
        uint256 _maxVolume,
        uint256 _minDeposit,
        uint256 _endTime,
        address _trader,
        address _stableSwap,
        address _curveLPToken,
        address _ldoRouter,
        address _gauge
    ) external {
        ...
        OptionsTrading trading = new OptionsTrading(_trader, governance);
        optionsTrading = payable(address(trading));
        ...
    }
```

comcat : Add receive function in the StakingVault contract

```
receive() external payable {
        require(msg.sender == STABLE_SWAP, "not allowed");
    }
```

jayphbee : implement `receive()` in `OptionTrading` and `StakingVault` contracts.

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.