Competitive Security Assessment

Savvy

Oct 2nd, 2023

Secure3

secure3.io

Summary 3

Overview 4

Audit Scope 5

Code Assessment Findings 6

SAV-1:Lack of permission check in the function `addTokenZapDetails()` 8

SAV-2:Malicious transfers make `zap()` function unable to success 12

SAV-3:The call of function `wrap()` may fail due of lack of `receive()` function 13

SAV-4:use __Ownable2Step_init() instead of __Ownable_init() 18

SAV-5:Gas Optimization: Cache array length outside of loop 19

SAV-6:Gas Optimization: Unnecessary set value to 0 21

SAV-7:Gas Optimization: Use different variable type or modifer 23

SAV-8:Unused imports 26

SAV-9:Gas Optimization: `++i` costs less gas than `i++`, especially when it's used in `for` loops 27

SAV-10:Unlocked Pragma Version 28

Disclaimer 29

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	Savvy
Platform & Language	Solidity
Codebase	• https://github.com/savvy-finance/savvy-contracts-internal • audit commit - 5d37828bba90c34f731d1337e362eddd8829b2cc • final commit - 88f832ba269ffd9791fc7af1442a6cdc34558682
Audit Methodology	• Audit Contest • Business Logic and Code Review • Privileged Roles Review • Static Analysis

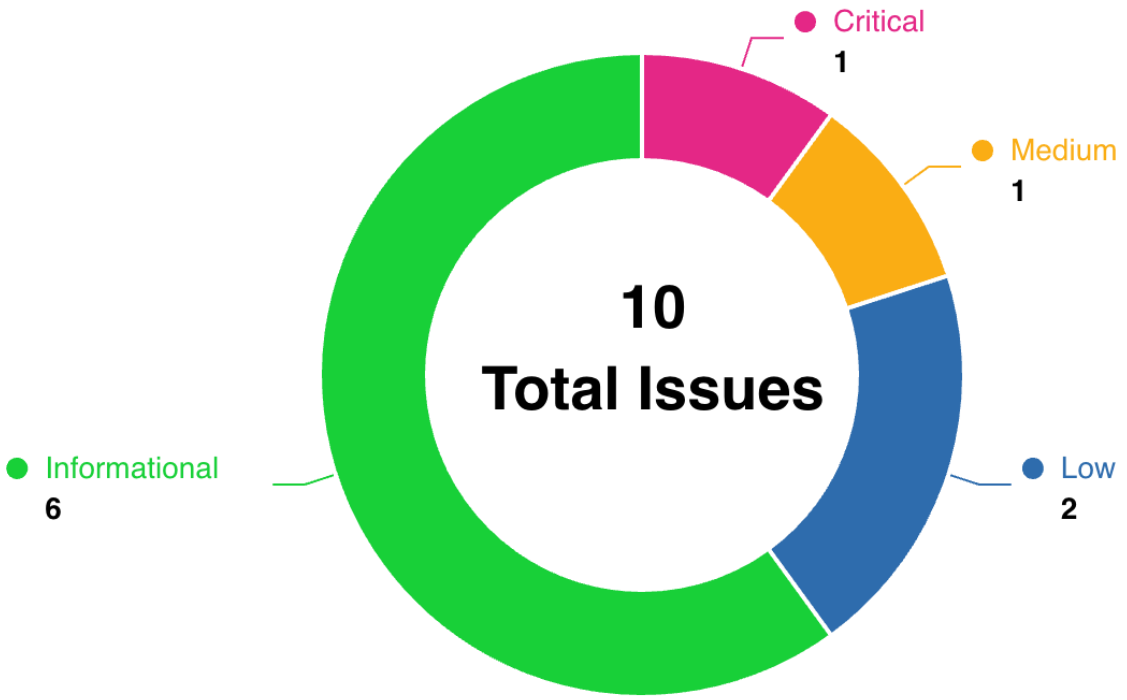
Code Vulnerability Review Summary

Vulnerability Level	Total	Reported	Acknowledged	Fixed	Mitigated	Declined
Critical	1	0	0	1	0	0
Medium	1	0	0	1	0	0
Low	2	0	0	1	0	1
Informational	6	0	0	6	0	0

Audit Scope

File	SHA256 Hash
contracts/SavvyIFOHelper.sol	5d37828bba90c34f731d1337e362eddd8829b2cc
contracts/adapters/gmd/GmdTokenAdapter.sol	5d37828bba90c34f731d1337e362eddd8829b2cc
contracts/adapters/beefy/BeefyZapperAdapter.sol	5d37828bba90c34f731d1337e362eddd8829b2cc

Code Assessment Findings



ID	Name	Category	Severity	Client Response	Contributor
SAV-1	Lack of permission check in the function `addTokenZapDetails()`	Logical	Critical	Fixed	Yaodao, rajatbeladiya, LiRiu
SAV-2	Malicious transfers make `zap()` function unable to success	Logical	Medium	Fixed	Yaodao
SAV-3	The call of function `wrap()` may fail due of lack of `receive()` function	Logical	Low	Fixed	Yaodao
SAV-4	use __Ownable2Step_init() instead of __Ownable_init()	Logical	Low	Declined	rajatbeladiya

SAV-5	Gas Optimization: Cache array length outside of loop	Gas Optimization	Informational	Fixed	rajatbeladiya, LiRiu
SAV-6	Gas Optimization: Unnecessary set value to 0	Gas Optimization	Informational	Fixed	Yaodao
SAV-7	Gas Optimization: Use different variable type or modifer	Code Style	Informational	Fixed	Yaodao, rajatbeladiya, LiRiu
SAV-8	Unused imports	Code Style	Informational	Fixed	Yaodao
SAV-9	Gas Optimization: `++i` costs less gas than `i++`, especially when it's used in `for` loops	Gas Optimization	Informational	Fixed	rajatbeladiya, LiRiu
SAV-10	Unlocked Pragma Version	Code Style	Informational	Fixed	Yaodao

SAV-1:Lack of permission check in the function `addTokenZapDetails()`

Category	Severity	Client Response	Contributor
Logical	Critical	Fixed	Yaodao, rajatbeladiya, LiRiu

Code Reference

- code/contracts/SavvyIFOHelper.sol#L44-L58
- code/contracts/SavvyIFOHelper.sol#L68-L76
- code/contracts/SavvyIFOHelper.sol#L121
- code/contracts/SavvyIFOHelper.sol#L174-L190
- code/contracts/SavvyIFOHelper.sol#L44

```
44:    function addTokenZapDetails(

44:    function addTokenZapDetails(
45:        address token,
46:        address spm,
47:        address strategy
48:    ) external {
49:        require(token != address(0), "token cannot be zero address");
50:        require(
51:            (spm != address(0) && strategy != address(0)) ||
52:                (spm == address(0) && strategy == address(0)),
53:            "must provide both SPM and strategy or neither"
54:        );
55:
56:        tokenToSPMMap[token] = spm;
57:        tokenToStrategyMap[token] = strategy;
58:    }

68:    function zap(
69:        address token,
70:        uint256 amount
71:    )
72:        external
73:        nonReentrant
74:        whenNotPaused
75:        returns (uint256 tokenOut, uint256 syntheticOut, address syntheticToken)
76:    {

121:        (tokenOut, syntheticOut) = _returnFunds(spm, token);

174:    function _returnFunds(
175:        ISavvyPositionManager spm,
176:        address token
177:    ) internal returns (uint256 tokenOut, uint256 syntheticOut) {
178:        address sender = msg.sender;
179:        tokenOut = _safeTransfer(token, sender);
180:        syntheticOut = _safeTransfer(spm.debtToken(), sender);
181:
182:        require(
183:            TokenUtils.safeBalanceOf(token, address(this)) == 0,
184:            "didn't transfer all base tokens"
185:        );
```

```
186:         require(
187:             TokenUtils.safeBalanceOf(spm.debtToken(), address(this)) == 0,
188:             "didn't transfer all synthetic tokens"
189:         );
190:     }
```

Description

Yaodao : The function `addTokenZapDetails()` can update the values of `tokenToSPMMap[token]` and `tokenToStrategyMap[token]`, but the function lacks of permission check. So these two variables can be updated by anyone. However, the value of `tokenToSPMMap[token]` is use for interface `ISavvyPositionManager` in other functions. As a result, the attacker can call `addTokenZapDetails()` to update the values of `tokenToSPMMap[token]` and `tokenToStrategyMap[token]` to be the attack contract addresses and then call the other functions to attack the protocol.

```
function addTokenZapDetails(
        address token,
        address spm,
        address strategy
    ) external {
        require(token != address(0), "token cannot be zero address");
        require(
            (spm != address(0) && strategy != address(0)) ||
                (spm == address(0) && strategy == address(0)),
            "must provide both SPM and strategy or neither"
        );

        tokenToSPMMap[token] = spm;
        tokenToStrategyMap[token] = strategy;
    }
```

rajatbeladiya : The contract `SavvyIFOHelper.sol` has two critical issues that can result in unauthorized fund transfers with two steps:

1. Lack of Access Control in `addTokenZapDetails()`
   - The function `addTokenZapDetails()` allows anyone to modify the mappings `tokenToSPMMap` and `tokenToStrategyMap`. This `tokenToSPMMap` and `tokenToStrategyMap` used in `zap()` function to initialize `ISavvyPositionManager` and `strategy`, it will lead to use malicious contract by `zap()` function . This lack of access control enables an attacker to associate tokens with their own malicious contracts.
2. Insufficient Input Validation in `zap()`
   - The `zap` function, which can be called by anyone, does not perform input validation for the token and amount parameters.
   - As explained in point 1 - This lack of input validation allows an attacker to supply manipulated or malicious token addresses and arbitrary amounts, it will initialize `ISavvyPositionManager` and `strategy` as malicious contract.

- `_returnFunds` is used internally in `zap()` to return funds to user from the contract. So, by using the `zap()` function, attacker can transfer funds using malicious token contract.

`Impact:` These issues can have severe consequences, as an attacker can associate their malicious contracts with tokens using `addTokenZapDetails()` and then execute the `zap()` function with manipulated token addresses. The subsequent operations in the `zap()` function, such as `_withdraw()`, and `_returnFunds()`, can transfer funds to the attacker's contracts instead of the intended recipient. This can result in significant financial loss for the contract and its users.

LiRiu : The `addTokenZapDetails` function lacks authentication and does not include proper checks for the non-empty values of `tokenToSPMMap[token]` and `tokenToStrategyMap[token]`.

This exposes a vulnerability where any user can overwrite `tokenToSPMMap` or `tokenToStrategyMap`.

In the event that `tokenToSPMMap` is overwritten with a malicious contract that always returns 0 for the `.balanceOf(address)` function, it would render all users unable to invoke the `zap` function.

Recommendation

Yaodao : Recommend adding the `onlyOwner` modifier.

rajatbeladiya : 1. add access control to `addTokenZapDetails()` function 2. perform input validation for `token` and `amount` in the `zap()` function, restrict use only whitelisted token

LiRiu : To add authentication to the addTokenZapDetails function, you can modify it by applying the `onlyOwner` modifier.

```
    function addTokenZapDetails(
        address token,
        address spm,
        address strategy
    ) external onlyOwner
```

Client Response

Fixed

SAV-2:Malicious transfers make `zap()` function unable to success

Category	Severity	Client Response	Contributor
Logical	Medium	Fixed	Yaodao

Code Reference

- code/contracts/SavvyIFOHelper.sol#L88-L91

```
88:         require(
89:             TokenUtils.safeBalanceOf(token, address(this)) == 0,
90:             "has unexpected base token balance"
91:         );
```

Description

Yaodao : In the function `zap()`, the balance of `token` will be checked whether it is zero. The attacker can transfer 1 wei token into the contract directly and then the check will revert. Besides, there is no withdraw function to withdraw these tokens and the check will always revert. As a result, the function `zap()` will never success.

```
    require(
        TokenUtils.safeBalanceOf(token, address(this)) == 0,
        "has unexpected base token balance"
    );
```

Recommendation

Yaodao : Recommend removing the check if it is not necessary or adding withdraw function to withdraw the malicious transfer tokens because it may still be DOS attacked.

Client Response

Fixed

SAV-3:The call of function `wrap()` may fail due of lack of `receive()` function

Category	Severity	Client Response	Contributor
Logical	Low	Fixed	Yaodao

Code Reference

- code/contracts/adapters/beefy/BeefyZapperAdapter.sol#L129-L145

```
129:    function _deposit(
130:        uint256 amount,
131:        address recipient
132:    ) internal returns (uint256) {
133:        uint256 balanceBefore = IERC20(token).balanceOf(address(this));
134:        (,uint256 swapAmountOut,) = beefyZapper.estimateSwap(token, baseToken, amount);
135:        // The minimum amount of the pair token the zapper should expect.
136:        // e.g. if the zapper receives 1000 USDC, then the minumum USDT it should
137:        // expect after swapping is ~490.
138:        uint256 pairTokenAmountOutMin = swapAmountOut * 99_000 / 100_000; // 1% slippage
139:        beefyZapper.beefIn(token, pairTokenAmountOutMin, baseToken, amount);
140:        uint256 balanceAfter = IERC20(token).balanceOf(address(this));
141:        uint256 receivedVaultTokens = balanceAfter - balanceBefore;
142:        TokenUtils.safeTransfer(token, recipient, receivedVaultTokens);
143:
144:        return receivedVaultTokens;
145:    }
```

Description

Yaodao : The function `_deposit()` will call the function `beefyZapper.beefIn()`. The function `beefyZapper.beefIn()` is not in the scope. According to the contract on-chain(https://arbiscan.io/address/0x29d9e421e4e4a7f151c00f7bff808a2b5f62f227#code), the function `beefIn()` will call function `_swapAndStake()`.

```
function beefIn (address beefyVault, uint256 tokenAmountOutMin, address tokenIn, uint256 tokenInAmou
nt) external {
        require(tokenInAmount >= minimumAmount, 'Beefy: Insignificant input amount');
        require(IERC20(tokenIn).allowance(msg.sender, address(this)) >= tokenInAmount, 'Beefy: Input
token is not approved');

        IERC20(tokenIn).safeTransferFrom(msg.sender, address(this), tokenInAmount);

        _swapAndStake(beefyVault, tokenAmountOutMin, tokenIn);
    }
```

In the function `_swapAndStake()`, it will swap first, then add LP and pledge LP to vault, then return the share of vault, and finally call `_returnAssets()`.

```
function _swapAndStake(address beefyVault, uint256 tokenAmountOutMin, address tokenIn) private {
        (IBeefyVaultV6 vault, IUniswapV2Pair pair) = _getVaultPair(beefyVault);

        (uint256 reserveA, uint256 reserveB,) = pair.getReserves();
        require(reserveA > minimumAmount && reserveB > minimumAmount, 'Beefy: Liquidity pair reserve
s too low');

        bool isInputA = pair.token0() == tokenIn;
        require(isInputA || pair.token1() == tokenIn, 'Beefy: Input token not present in liqudity pa
ir');

        address[] memory path = new address[](2);
        path[0] = tokenIn;
        path[1] = isInputA ? pair.token1() : pair.token0();

        uint256 fullInvestment = IERC20(tokenIn).balanceOf(address(this));
        uint256 swapAmountIn;
        if (isInputA) {
            swapAmountIn = _getSwapAmount(pair, fullInvestment, reserveA, reserveB, path[0], path
[1]);
        } else {
            swapAmountIn = _getSwapAmount(pair, fullInvestment, reserveB, reserveA, path[0], path
[1]);
        }

        _approveTokenIfNeeded(path[0], address(router));
        uint256[] memory swapedAmounts = router
            .swapExactTokensForTokensSimple(swapAmountIn, tokenAmountOutMin, path[0], path[1], pair.
stable(), address(this), block.timestamp);

        _approveTokenIfNeeded(path[1], address(router));
        (,, uint256 amountLiquidity) = router
            .addLiquidity(path[0], path[1], pair.stable(), fullInvestment.sub(swapedAmounts[0]), swa
pedAmounts[1], 1, 1, address(this), block.timestamp);

        _approveTokenIfNeeded(address(pair), address(vault));
        vault.deposit(amountLiquidity);

        vault.safeTransfer(msg.sender, vault.balanceOf(address(this)));
        _returnAssets(path);
    }
```

In the function `_returnAssets()`, it will refund the tokens to the `msg.sender`, which is `BeefyZapperAdapter` here. However, there is no logic to deal with these refund tokens.

```
function _returnAssets(address[] memory tokens) private {
        uint256 balance;
        for (uint256 i; i < tokens.length; i++) {
            balance = IERC20(tokens[i]).balanceOf(address(this));
            if (balance > 0) {
                if (tokens[i] == WETH) {
                    IWETH(WETH).withdraw(balance);
                    (bool success,) = msg.sender.call{value: balance}(new bytes(0));
                    require(success, 'Beefy: ETH transfer failed');
                } else {
                    IERC20(tokens[i]).safeTransfer(msg.sender, balance);
                }
            }
        }
    }
```

Yaodao : The function `_deposit()` call by `warp()` will call the function `beefyZapper.beefIn()`. The function `beefyZapper.beefIn()` is not in the scope. According to the contract on-chain(https://arbiscan.io/address/0x29d9e421e4e4a7f151c00f7bff808a2b5f62f227#code), the function `beefIn()` will call function `_swapAndStake()`.

```
function beefIn (address beefyVault, uint256 tokenAmountOutMin, address tokenIn, uint256 tokenInAmou
nt) external {
        require(tokenInAmount >= minimumAmount, 'Beefy: Insignificant input amount');
        require(IERC20(tokenIn).allowance(msg.sender, address(this)) >= tokenInAmount, 'Beefy: Input
token is not approved');

        IERC20(tokenIn).safeTransferFrom(msg.sender, address(this), tokenInAmount);

        _swapAndStake(beefyVault, tokenAmountOutMin, tokenIn);
    }
```

In the function `_swapAndStake()`, it will swap first, then add LP and pledge LP to vault, then return the share of vault, and finally call `_returnAssets()`.

In the function `_returnAssets()`, it will refund the tokens to the `msg.sender`, which is `BeefyZapperAdapter` here.

```
function _returnAssets(address[] memory tokens) private {
        uint256 balance;
        for (uint256 i; i < tokens.length; i++) {
            balance = IERC20(tokens[i]).balanceOf(address(this));
            if (balance > 0) {
                if (tokens[i] == WETH) {
                    IWETH(WETH).withdraw(balance);
                    (bool success,) = msg.sender.call{value: balance}(new bytes(0));
                    require(success, 'Beefy: ETH transfer failed');
                } else {
                    IERC20(tokens[i]).safeTransfer(msg.sender, balance);
                }
            }
        }
    }
```

Besides, in the function `_returnAssets()`, it will convert `WETH` to `ETH` and then transfer `ETH` to the contract `BeefyZapperAdapter`. However, there is no `receive()` function in the contract `BeefyZapperAdapter`. As a result, the call of `wrap()` will fail.

Recommendation

Yaodao : Recommend adding the logic to deal with the refund tokens.
Yaodao : Recommend adding `receive()` function in the contract `BeefyZapperAdapter`.

Client Response

Fixed

SAV-4:use __Ownable2Step_init() instead of __Ownable_init()

Category	Severity	Client Response	Contributor
Logical	Low	Declined	rajatbeladiya

Code Reference

- code/contracts/SavvyIFOHelper.sol#L32
- code/contracts/adapters/beefy/BeefyZapperAdapter.sol#L50
- code/contracts/adapters/gmd/GmdTokenAdapter.sol#L74

```
32:         __Ownable_init();

50:         __Ownable2Step_init();

74:         __Ownable2Step_init();
```

Description

rajatbeladiya : `SavvyIFOHelper.sol` used `__Ownable_init()` in the `initializer()` to initiate owner, but `BeefyZapperAdapter.sol` and `GmdTokenAdapter.sol` used `__Ownable2Step_init()` to initialize owner with 2 step. It means `SavvyIFOHelper.sol` cannot change the owner once it initialized.

`Impact:` ownership can be lost if the ownership transferred to non accessible address.

Recommendation

rajatbeladiya : use `__Ownable2Step_init()` instead of the `__Ownable_init()` in the `SavvyIFOHelper.sol`

Client Response

Declined, In "@openzeppelin/contracts-upgradeable": "^4.8.2", __Ownable2Step_init() and __Ownable_init() are implemented the same way. They set the first owner of the contract to the msg.sender. In a Ownable2Step contract there is no need for the deployer of the contract to approve ownership of the contract. Its redundant. Furthermore, this is no impact on Ownable2Step to work as intended. We've validated this by transferring ownership of our YieldStrategyManager which extends Ownable2StepUpgradeable and calls __Ownable_init().

The rationale for this issue is incorrect and therefore the issue should be removed from the findings.

__Ownable_init() implementation https://github.com/OpenZeppelin/openzeppelin-contracts-upgradeable/blob/release-v4.8/contracts/access/OwnableUpgradeable.sol#L30

__Ownable2Step_init() https://github.com/OpenZeppelin/openzeppelin-contracts-upgradeable/blob/release-v4.8/contracts/access/Ownable2StepUpgradeable.sol#L22

SAV-5:Gas Optimization: Cache array length outside of loop

Category	Severity	Client Response	Contributor
Gas Optimization	Informational	Fixed	rajatbeladiya, LiRiu

Code Reference

- code/contracts/SavvyIFOHelper.sol#L106
- code/contracts/adapters/beefy/BeefyZapperAdapter.sol#L64
- code/contracts/adapters/gmd/GmdTokenAdapter.sol#L92
- code/contracts/SavvyIFOHelper.sol#L107

```
64:         for (uint256 i = 0; i < allowlistAddresses.length; i++) {

92:         for (uint256 i = 0; i < allowlistAddresses.length; i++) {

106:         for (uint256 i = 0; i < maxBorrowBySPM.length && maxBorrowAmount == 0; i++) {

107:             if (maxBorrowBySPM[i].baseToken == tokenToSPMMap[token]) {
```

Description

rajatbeladiya : If not cached, the solidity compiler will always read the length of the array during each iteration. That is, if it is a storage array, this is an extra sload operation (100 additional extra gas for each iteration except for the first) and if it is a memory array, this is an extra mload operation (3 additional gas for each iteration except for the first).
LiRiu : State variables should be cached in stack variables rather than re-reading them from storage

Recommendation

rajatbeladiya : Cache array length outside of loop

```
uint256 maxBorrowBySPMLength = maxBorrowBySPM.length;
for (uint256 i = 0; i < maxBorrowBySPMLength && maxBorrowAmount == 0; i++) {
            if (maxBorrowBySPM[i].baseToken == tokenToSPMMap[token]) {
                maxBorrowAmount = maxBorrowBySPM[i].amount;
            }
        }
```

LiRiu : You just need to replace `tokenToSPMMap[token]` with `address(spm)` in the following line: `if (maxBorrowBySPM[i].baseToken == tokenToSPMMap[token]) {`

Client Response

Fixed, rajatbeladiya explanation was exceptionally helpful

SAV-6:Gas Optimization: Unnecessary set value to 0

Category	Severity	Client Response	Contributor
Gas Optimization	Informational	Fixed	Yaodao

Code Reference

- code/contracts/adapters/gmd/GmdTokenAdapter.sol#L157

```
157:        uint256 withdrawnAmount = 0;
```

Description

Yaodao : Since all default values in solidity are already 0, it is unnecessary to initialize `withdrawnAmount` as 0 in function `_withdraw`:

```
function _withdraw(
        uint256 amount,
        address recipient
    ) internal returns (uint256, uint256) {
        _checkPoolId();

        uint256 withdrawnAmount = 0;
        ...
    }
```

Recommendation

Yaodao : Recommend following fix to save gas:

```
function _withdraw(
        uint256 amount,
        address recipient
    ) internal returns (uint256, uint256) {
        _checkPoolId();

        uint256 withdrawnAmount;
        ...
    }
```

Client Response

Fixed

SAV-7:Gas Optimization: Use different variable type or modifer

Category	Severity	Client Response	Contributor
Code Style	Informational	Fixed	Yaodao, rajatbeladiya, LiRiu

Code Reference

- code/contracts/adapters/gmd/GmdTokenAdapter.sol#L23
- code/contracts/adapters/beefy/BeefyZapperAdapter.sol#L23
- code/contracts/adapters/beefy/BeefyZapperAdapter.sol#L59-L67
- code/contracts/adapters/gmd/GmdTokenAdapter.sol#L87-L95
- code/contracts/adapters/beefy/BeefyZapperAdapter.sol#L60
- code/contracts/adapters/gmd/GmdTokenAdapter.sol#L88

```
23:    string public constant override version = "1.0.0";

23:    string public constant override version = "1.0.0";

59:    function addAllowlist(
60:        address[] memory allowlistAddresses,
61:        bool status
62:    ) external override onlyOwner {
63:        require(allowlistAddresses.length > 0, "invalid length");
64:        for (uint256 i = 0; i < allowlistAddresses.length; i++) {
65:            isAllowlisted[allowlistAddresses[i]] = status;
66:        }
67:    }

60:        address[] memory allowlistAddresses,

87:    function addAllowlist(
88:        address[] memory allowlistAddresses,
89:        bool status
90:    ) external override onlyOwner {
91:        require(allowlistAddresses.length > 0, "invalid length");
92:        for (uint256 i = 0; i < allowlistAddresses.length; i++) {
93:            isAllowlisted[allowlistAddresses[i]] = status;
94:        }
95:    }

88:        address[] memory allowlistAddresses,
```

Description

Yaodao : It's better to use `calldata` instead of memory for function parameters that represent variables that will not be modified.
rajatbeladiya : Mark data types as `calldata` instead of `memory` where possible. This makes it so that the data is not automatically loaded into memory. If the data passed into the function does not need to be changed (like updating values in an array), it can be passed in as `calldata`. The one exception to this is if the argument must later be passed into another function that takes an argument that specifies `memory` storage.
LiRiu : Using the 'private' modifier to declare global constants can reduce gas consumption.

Recommendation

Yaodao : Recommend using calldata instead of memory to save gas.

```
function addAllowlist(
        address[] calldata allowlistAddresses,
        bool status
    ) external override onlyOwner {
        require(allowlistAddresses.length > 0, "invalid length");
        for (uint256 i = 0; i < allowlistAddresses.length; i++) {
            isAllowlisted[allowlistAddresses[i]] = status;
        }
    }
```

rajatbeladiya : Use `calldata` instead of `memory`

LiRiu : Using the 'private' modifier to declare global constants

```
string private constant override version = "1.0.0";
```

Client Response

Fixed, We are declining LiRiu's private modifier as we want the versions to be publicly accessible.

SAV-8:Unused imports

Category	Severity	Client Response	Contributor
Code Style	Informational	Fixed	Yaodao

Code Reference

- code/contracts/adapters/beefy/BeefyZapperAdapter.sol#L4
- code/contracts/adapters/gmd/GmdTokenAdapter.sol#L4-L6

```
4:import {IllegalState} from "../../base/Errors.sol";

4:import {IllegalState} from "../../base/Errors.sol";
5:
6:import "@openzeppelin/contracts/token/ERC20/IERC20.sol";
```

Description

Yaodao : The contract `BeefyZapperAdapter` includes the following unused imports:

```
import {IllegalState} from "../../base/Errors.sol";
```

The contract `GmdTokenAdapter` includes the following unused imports:

```
import {IllegalState} from "../../base/Errors.sol";
import "@openzeppelin/contracts/token/ERC20/IERC20.sol";
```

Recommendation

Yaodao : Recommend removing the import statement to save on deployment gas costs.

Client Response

Fixed

SAV-9:Gas Optimization: `++i` costs less gas than `i++`, especially when it's used in `for` loops

Category	Severity	Client Response	Contributor
Gas Optimization	Informational	Fixed	rajatbeladiya, LiRiu

Code Reference

- code/contracts/SavvyIFOHelper.sol#L106
- code/contracts/adapters/beefy/BeefyZapperAdapter.sol#L64
- code/contracts/adapters/gmd/GmdTokenAdapter.sol#L92
- code/contracts/SavvyIFOHelper.sol#L106

```
64 :        for (uint256 i = 0; i < allowlistAddresses.length; i++) {

92:        for (uint256 i = 0; i < allowlistAddresses.length; i++) {

106:        for (uint256 i = 0; i < maxBorrowBySPM.length && maxBorrowAmount == 0; i++) {

106 :        for (uint256 i = 0; i < maxBorrowBySPM.length && maxBorrowAmount == 0; i++) {
```

Description

rajatbeladiya : It saves 5 gas per loop by using `++i` instead of `i++`, especially when it's used in `for` loops (`--i` / `i--` too).
LiRiu : `++i` costs less gas than `i++`, especially when it's used in for-loops (`--i` / `i--` too)

Recommendation

rajatbeladiya : use `++i` instead of `i++`
LiRiu : Recommend using `++i` instead of `i++`, `--i` / `i--` too

Client Response

Fixed

SAV-10:Unlocked Pragma Version

Category	Severity	Client Response	Contributor
Code Style	Informational	Fixed	Yaodao

Code Reference

- code/contracts/SavvyIFOHelper.sol#L2
- code/contracts/adapters/beefy/BeefyZapperAdapter.sol#L2
- code/contracts/adapters/gmd/GmdTokenAdapter.sol#L2

```
2:pragma solidity ^0.8.17;

2:pragma solidity ^0.8.17;

2:pragma solidity ^0.8.17;
```

Description

Yaodao : Solidity files in packages have a pragma version `^0.8.17`. The caret `(^)` points to unlocked pragma, meaning the compiler will use the specified version or above.

Recommendation

Yaodao : Recommend the compiler version is instead locked at the lowest version possible that the contract can be compiled at.

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.