Competitive Security Assessment

Apebond

Dec 7th, 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

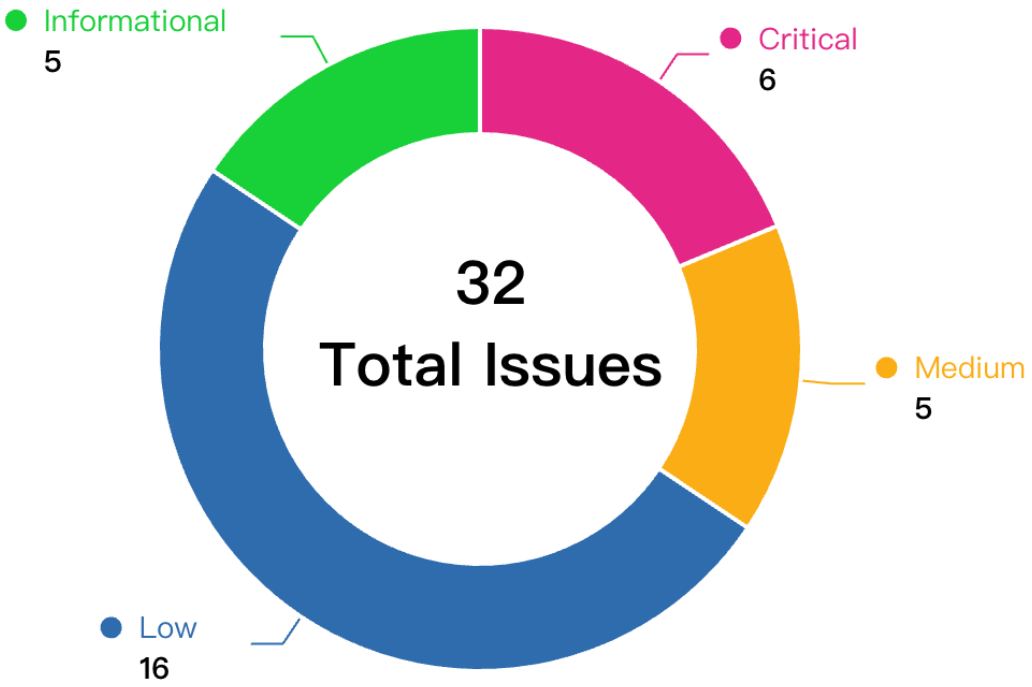
Project Name	Apebond
Platform & Language	Solidity
Codebase	• https://github.com/SoulSolidity/SoulZapV1 • audit commit - 158665422506e5e11e53a1801637fa656a42d3bb • final commit - 20f069cfb671c5af02958c1b3f33377d085fa659
Audit Methodology	• Audit Contest • Business Logic and Code Review • Privileged Roles Review • Static Analysis

Audit Scope

File	SHA256 Hash
./contracts/SoulZap_UniV2_Lens.sol	67d4ec44bafcd7d865024aafb2b5f1512ee6b1712e2b592231eb92e8e9926458
./contracts/SoulZap_UniV2.sol	c6b3ef1a392dcc7f49f3dbc2f6928b0081f9fa6e813bbda023a89178d8495a05
./contracts/extensions/ApeBond/lib/ICustomBillRefillable.sol	6e5a97877859dbfe9361b12913db805a9795ecd5c715a73a2087545bad836f2e
./contracts/extensions/Arrakis/lib/ArrakisMath.sol	7db2aad7c83369162b3a5c4d2c9102f3b8e933ec1bd083d1e7adf4371f47be22
./contracts/fee-manager/SoulFeeManager.sol	3614e01105bcdeb1dd1d76a568605289135469e198be41305a33c91529408977
./contracts/extensions/ApeBond/SoulZap_Ext_ApeBond_Lens.sol	cc28c6addaa3f546a05618d44ab8b15e6644581864c2e588bf7bbeca446febb7
./contracts/utils/EpochVolumeTracker.sol	acd1d42ecf40008c5ba0ff736e699216699502dd2081fa1f6c68f39135b48df0
./contracts/extensions/ApeBond/SoulZap_Ext_ApeBond.sol	76b91aabce1d953b661019931dc5ed32538c26a239e9d24ea54aa2c05fb10881
./contracts/ISoulZap_UniV2.sol	1c92d54193aa714c4ef151a17b558f615cf294aae025a7af829016b643e23f68
./contracts/extensions/Arrakis/lib/IArrakisRouter.sol	41c925c1d95e2b5cee2c8a1bc4e9169bef07cff6e64b2c0fbe9eff42b68c28da
./contracts/access/SoulAccessRegistry.sol	232dfea24e9600ed40bcf5e235d315b63d1da1276cdd9e04cc06bf29642f93ac
./contracts/utils/TransferHelper.sol	9a984a986035a33fc566d7102e5aeddc883636f04ad04d395349d379bb75688b
./contracts/extensions/ApeBond/lib/IMasterApe.sol	39e34077b91ab0685332b7014068a64df654fa8ef69ba7085fb87adfc5017dd5
./contracts/extensions/ApeBond/SoulZap_Ext_BondNftWhitelist.sol	34c06107563669df7f9e6fc9d17b1af8210eb41bbfff97cd9cfe5f0c6468b7e9
./contracts/access/SoulAccessManaged.sol	b48042bad90c4e1bf5559f727ac5381384e1130b7e167bd67bb1f242fd725315

./contracts/extensions/Arrakis/lib/IArrakisPool.sol	6e2b2dcb81c30457c002fa74c5ff14d0469be5811ffd75c826e3abe023a8d6c0
./contracts/extensions/Arrakis/lib/IArrakisFactoryV1.sol	89ea444c7c23fb6e687345d30c1fba7b41b372e5de413940a519acc061ee0791
./contracts/fee-manager/ISoulFeeManager.sol	56f95afef62ccf566b21e32ed3f6c4c90a98538c74276ba22d71c124d8fba993
./contracts/utils/IEpochVolumeTracker.sol	90618d42eb721bdc6bb28cd5cef31b32f2aa6335c0d2ef0626886147eb3cbdfe
./contracts/full-versions/SoulZap_UniV2_Extended_V1.sol	e8d5cc9d3449b614d7212e34fa17ad3211bc034c947ce35f2ad2a79e6741cd5a
./contracts/full-versions/SoulZap_UniV2_Extended_V1_Lens.sol	cd78733c39a16ce5a4b5d52b02c1b615abffb91b652be2cdf7c4ae798682dc12
./contracts/utils/LocalVarsLib.sol	222c68658e27b9a18c15eb5cb64bb51b614704dd81e6be9ec56a5489ae1d24c4
./contracts/extensions/ApeBond/lib/IPoolManager.sol	d184d1b50f1fb90410bffb85f28f9fb11ea4fb13d3a7111e8cc61e5434abc767
./contracts/lib/IWETH.sol	98432e91ce293d043783520c6e47ebb28590c0ac7b9c9617b698cffcf326e427
./contracts/utils/Constants.sol	bc3035051cb735517db6abbb5edb5f4d7d815f0f022e0c011eab6bfb851eb130
./contracts/utils/ITransferHelper.sol	207c4c4c12c6b52f98fe0a8ee5287a5c378a672442a3f6f426cbd6e5f980fcdb
./contracts/access/ISoulAccessManaged.sol	ceb56f36d40375476ce73fac1778a426630a8363ff7f8719ad4658cc07356d56
./contracts/extensions/ApeBond/lib/IUniV3PriceGetter.sol	8adadff2239d7ddfa679a6848b180d4bf675f4ace8471427fa43e1739353fd8c

Code Assessment Findings



ID	Name	Category	Severity	Client Response	Contributor
APB-1	Potential risk of sandwich attack in `zap()`	Logical	Critical	Fixed	Yaodao
APB-2	Approve USDT should approve 0 first	Logical	Critical	Fixed	Yaodao, SerSomeone, parth_15, Kong7ych3, jayphbee
APB-3	Taking fee from user Incorrectly in `zapBond()`	Logical	Critical	Fixed	Yaodao

APB-4	`SoulZap_UniV2._zap()` function has approval to router contract even after completion of transaction	Privilege Related	Critical	Fixed	parth_15
APB-5	Underflow error in function `SoulFeeManager.getFee()`	Integer Overflow and Underflow	Critical	Fixed	parth_15, TrungOre
APB-6	Missing check `msg.value == 0` in `SoulZap::verifyMsgValueAndWrap`	Logical	Critical	Fixed	infinityhacker
APB-7	Lower bound fee rate for `SoulFeeManager::getFee()` is incorrect	Logical	Medium	Fixed	ravikiran_web3
APB-8	ArrakisMath inconsistencies for different tokens decimals	Integer Overflow and Underflow	Medium	Fixed	ginlee, ravikiran_web3, BradMoonUESTC, jayphbee
APB-9	Inaccurate Token Amount Calculation in Liquidity Addition	Logical	Medium	Acknowledged	parth_15, BradMoonUESTC
APB-10	Lack of checking actual tokens received (Deflationary token)	Logical	Medium	Fixed	Kong7ych3
APB-11	Addressing Duplicate Addresses in Token Swap Paths for Enhanced Security and Efficiency	Logical	Medium	Fixed	TrungOre, BradMoonUESTC
APB-12	Unwithdrawable tokens permanently locked in the contract	Logical	Low	Fixed	Yaodao
APB-13	No Protection of Uninitialized Implementation Contracts From Attacker in SoulAccessRegistry	Logical	Low	Fixed	parth_15
APB-14	lack of validation on caller provided `SwapPath.swapRouter` with fake router address	Logical	Low	Fixed	Yaodao, parth_15, Kong7ych3, TrungOre
APB-15	Uncheck return value of `token.approve()`	Logical	Low	Acknowledged	Yaodao, ginlee

APB-16	Big deadline used for swap/zap functions	Logical	Low	Fixed	parth_15, Chinmay
APB-17	Divide before multiply	Logical	Low	Fixed	Yaodao
APB-18	Any role can be set as Admin in SoulAccessRegistry::setRoleAdminByName()	Logical	Low	Fixed	ravikiran_web3
APB-19	Hardcoded gas value can cause DOS	Language Specific	Low	Acknowledged	parth_15
APB-20	Use abi.encodeCall instead of abi.encodeWithSelector in SoulZap_UniV2_Lens::getSwapData function	Language Specific	Low	Fixed	ginlee
APB-21	Wrong initialization of EpochVolumeTracker constructor will lead to very large epoch duration	Language Specific	Low	Fixed	parth_15, Kong7ych3, Chinmay
APB-22	Length of addressSet should be checked SoulFeeManager::getFeeToken	Language Specific	Low	Fixed	ravikiran_web3
APB-23	Don't use block.timestamp as deadline in SoulZap_UniV2_Lens::_getSwapData function	Logical	Low	Fixed	ginlee
APB-24	values returned by EpochVolumeTracker::getEpochVolumeInfo() are incorrect, buggy implementation	Logical	Low	Fixed	ravikiran_web3
APB-25	Project may fail to be deployed to chains not compatible with Shanghai hardfork	Language Specific	Low	Fixed	ginlee
APB-26	Absence of validation for feeSwapPath.swapRouter in the function SoulZap_UniV2._handleFee()	Logical	Low	Fixed	Kong7ych3
APB-27	Wrong comparison between amounts of tokens in _getFeeSwapPath function	Logical	Low	Fixed	Chinmay

APB-28	Repeated calls of modifier on internal functions	Gas Optimization	Informational	Fixed	Yaodao
APB-29	Use of `slot0` to get `sqrtPriceLimitX96` in ArrakisMath::pairTokensAndValue function can lead to price manipulation	Oracle Manipulation	Informational	Acknowledged	ginlee, ravikiran_web3, BradMoonUESTC
APB-30	modifier whenNotPaused can be omitted in _swap function	Language Specific	Informational	Fixed	parth_15
APB-31	Empty constructor not needed in abstract contract	Language Specific	Informational	Fixed	parth_15
APB-32	SoulFeeManager::isSoulFeeManager can be constant	Gas Optimization	Informational	Acknowledged	ravikiran_web3

APB-1: Potential risk of sandwich attack in `zap()`

Category	Severity	Client Response	Contributor
Logical	Critical	Fixed	Yaodao

Code Reference

- code/contracts/SoulZap_UniV2.sol#L250-L254
- code/contracts/SoulZap_UniV2.sol#L315-L329

```
250:// Ensure token addresses and paths are in ascending numerical order
251:        if (zapParams.token1 < zapParams.token0) {
252:            (zapParams.token0, zapParams.token1) = (zapParams.token1, zapParams.token0);
253:            (zapParams.path0, zapParams.path1) = (zapParams.path1, zapParams.path0);
254:        }

315:if (zapParams.liquidityPath.lpType == LPType.V2) {
316:            // Add liquidity to UniswapV2 Pool
317:            (vars.amount0Lp, vars.amount1Lp, ) = IUniswapV2Router02(zapParams.liquidityPath.lpRo
uter).addLiquidity(
318:                zapParams.token0,
319:                zapParams.token1,
320:                vars.amount0Out,
321:                vars.amount1Out,
322:                zapParams.liquidityPath.minAmountLP0,
323:                zapParams.liquidityPath.minAmountLP1,
324:                zapParams.to,
325:                zapParams.deadline
326:            );
327:        } else {
328:            revert("SoulZap: lpType not supported");
329:        }
```

Description

Yaodao : In the function `zap()`, the logic of the following codes ensures token addresses and paths are in ascending numerical order. The values `token0` and `token1`, `path0` and `path1` are updated.

```
        // Ensure token addresses and paths are in ascending numerical order
        if (zapParams.token1 < zapParams.token0) {
            (zapParams.token0, zapParams.token1) = (zapParams.token1, zapParams.token0);
            (zapParams.path0, zapParams.path1) = (zapParams.path1, zapParams.path0);
        }
```

However, the `zapParams.liquidityPath.minAmountLP0` and `zapParams.liquidityPath.minAmountLP1` corresponding to the previous `token0`, `token1` are used directly regardless of the update of `token0`, `token1`. As a result, the parameters may be inconsistent.

```
        if (zapParams.liquidityPath.lpType == LPType.V2) {
            // Add liquidity to UniswapV2 Pool
            (vars.amount0Lp, vars.amount1Lp, ) = IUniswapV2Router02(zapParams.liquidityPath.lpRoute
r).addLiquidity(
                zapParams.token0,
                zapParams.token1,
                vars.amount0Out,
                vars.amount1Out,
                zapParams.liquidityPath.minAmountLP0,
                zapParams.liquidityPath.minAmountLP1,
                zapParams.to,
                zapParams.deadline
            );
        } else {
            revert("SoulZap: lpType not supported");
        }
```

Since different tokens will have different decimals, the values of `zapParams.liquidityPath.minAmountLP0` and `zapParams.liquidityPath.minAmountLP1` may be quite different. For example, the token0 is USDT with a decimal of 6, and the token1 is WETH with a decimal of 18. Once the logic `zapParams.token1 < zapParams.token0` is triggerred, the values of `zapParams.liquidityPath.minAmountLP0` and `zapParams.liquidityPath.minAmountLP1` are incorrect and may cause the call to fail or be attacked by the sandwich attack due to the slip.

Recommendation

Yaodao : Recommend fixing the logic to update the `zapParams.liquidityPath.minAmountLP0` and `zapParams.liquidityPath.minAmountLP1` synchronizedly.

Client Response

Fixed. https://github.com/SoulSolidity/SoulZapV1/commit/1fba29e2d4d0c56017585bc44033debb7ffe904b removed code that initially ensured that token addresses and paths are in ascending numerical order because it's not necessary anymore (it was in old code)

APB-2:Approve USDT should approve 0 first

Category	Severity	Client Response	Contributor
Logical	**Critical**	**Fixed**	**Yaodao, SerSomeone, parth_15, Kong7ych3, jayphbee**

Code Reference

- code/contracts/SoulZap_UniV2.sol#L197
- code/contracts/SoulZap_UniV2.sol#L288
- code/contracts/SoulZap_UniV2.sol#L303
- code/contracts/SoulZap_UniV2.sol#L312-L329
- code/contracts/SoulZap_UniV2.sol#L312-L313
- code/contracts/SoulZap_UniV2.sol#L374
- code/contracts/SoulZap_UniV2.sol#L442

```
197:swapParams.tokenIn.approve(swapParams.path.swapRouter, swapParams.amountIn);

288:zapParams.tokenIn.approve(zapParams.path0.swapRouter, vars.amount0In);

303:zapParams.tokenIn.approve(zapParams.path1.swapRouter, vars.amount1In);

312:IERC20(zapParams.token0).approve(address(zapParams.liquidityPath.lpRouter), vars.amount0Out);
313:        IERC20(zapParams.token1).approve(address(zapParams.liquidityPath.lpRouter), vars.amount1
Out);
314:
315:        if (zapParams.liquidityPath.lpType == LPType.V2) {
316:            // Add liquidity to UniswapV2 Pool
317:            (vars.amount0Lp, vars.amount1Lp, ) = IUniswapV2Router02(zapParams.liquidityPath.lpRo
uter).addLiquidity(
318:                zapParams.token0,
319:                zapParams.token1,
320:                vars.amount0Out,
321:                vars.amount1Out,
322:                zapParams.liquidityPath.minAmountLP0,
323:                zapParams.liquidityPath.minAmountLP1,
324:                zapParams.to,
325:                zapParams.deadline
326:            );
327:        } else {
328:            revert("SoulZap: lpType not supported");
329:        }

312:IERC20(zapParams.token0).approve(address(zapParams.liquidityPath.lpRouter), vars.amount0Out);
313:        IERC20(zapParams.token1).approve(address(zapParams.liquidityPath.lpRouter), vars.amount1
Out);

374:IUniswapV2Router02(router).swapExactTokensForTokens(amountIn, amountOutMin, path, _to, deadlin
e);

442:_inputToken.approve(_feeSwapPath.swapRouter, inputFeeAmount);
```

Description

Yaodao : According to the codes, the function `zap()` is used to zap `the` single token to LP. The amounts of `token0` and `token1` will be calculated and then approve to the `zapParams.liquidityPath.lpRouter` to call the function

`addLiquidity()` of the router.

```
        IERC20(zapParams.token0).approve(address(zapParams.liquidityPath.lpRouter), vars.amount0Ou
t);
        IERC20(zapParams.token1).approve(address(zapParams.liquidityPath.lpRouter), vars.amount1Ou
t);

        if (zapParams.liquidityPath.lpType == LPType.V2) {
            // Add liquidity to UniswapV2 Pool
            (vars.amount0Lp, vars.amount1Lp, ) = IUniswapV2Router02(zapParams.liquidityPath.lpRoute
r).addLiquidity(
                zapParams.token0,
                zapParams.token1,
                vars.amount0Out,
                vars.amount1Out,
                zapParams.liquidityPath.minAmountLP0,
                zapParams.liquidityPath.minAmountLP1,
                zapParams.to,
                zapParams.deadline
            );
        } else {
            revert("SoulZap: lpType not supported");
        }
```

In the function `addLiquidity()` of UniswapV2, the amounts of `token0` and `token1` transferred to the router will be calculated again in the logic of `addLiquidity()`. As a result, the amounts transferred do not correspond to the approve amounts, and the allowance will not be fully used.

The `approve()` function of some ERC20 tokens can't be called when the current allowance is not 0. For example, the USDT token's `approve()` can't be called to approve a new amout(not 0) when the previous allowance is not fully used to be 0. As a result, the call of `zap()` will always fail because the approve of `USDT` will revert.

```
    /**
    * @dev Approve the passed address to spend the specified amount of tokens on behalf of msg.sende
r.
    * @param _spender The address which will spend the funds.
    * @param _value The amount of tokens to be spent.
    */
    function approve(address _spender, uint _value) public onlyPayloadSize(2 * 32) {

        // To change the approve amount you first have to reduce the addresses`
        //  allowance to zero by calling `approve(_spender, 0)` if it is not
        //  already 0 to mitigate the race condition described here:
        //  https://github.com/ethereum/EIPs/issues/20#issuecomment-263524729
        require(!((_value != 0) && (allowed[msg.sender][_spender] != 0)));

        allowed[msg.sender][_spender] = _value;
        Approval(msg.sender, _spender, _value);
    }
```

SerSomeone : There are good efforts to support fee-on-transfer tokens by checking the difference in balance between transfers to the `SoulZap_UniV2` contract.

However - the call to UniswapV2 router `swapExactTokensForTokens` will revert if fee-on-transfer tokens are used. The client should call `swapExactTokensForTokensSupportingFeeOnTransferTokens` instead which handles both fee-on-transfer tokens and regular tokens

Its important to note that popular tokens such as `USDT` and `USDC` could enable fee-on-transfer behaviour at any time. `USDT` has the implemented it in their contract, and `USDC` could be updated to include it.

parth_15 : The above calls approve that do not handle some tokens that are designed to mitigate approval race condition bug. Some token contracts revert the transaction when the allowance is not zero.

USDT may be a classic example for this.

Here's a short POC.

```
usdt.functions.approve(basket.address, 100).transact()
## the second tx would be reverted as the allowance is not zero
usdt.functions.approve(basket.address, 50).transact()
```

Kong7ych3 : In SoulZap_UniV2 contract, user can add token0/token1 liquidity to Uniswap v2 via zap function. The token0 and token1 tokens will be approved to `zapParams.liquidityPath.lpRouter` before the addLiquidity operation is performed, so that the lpRouter can successfully transfer tokens from the SoulZap_UniV2 contract. Note that the amount of liquidity added to the lpRouter contract is not exactly equal to the amount approved, so the zap function returns the remaining tokens to the user at the end.

This process is correct for normal ERC20 tokens, but will not work for tokens with weird approval function implementations (e.g. USDT). Their approval function must require that the original allowance must be equal to 0 when the approval amount is greater than 0. Therefore, in the zap function, after the first user performs an addLiquidity operation and there are USDT tokens remaining, the second user will REVERT during the zap operation at the step of

approving it to the lpRouter. because at this point, after the first user has used the SoulZap_UniV2 contract's remaining allowance to the lpRouter is not 0, which will result in all subsequent users adding USDT liquidity not being able to zap successfully.

Note that USDT in the example is a well known and widely used stablecoin, so it is necessary to be compatible with this strange ERC20 token.

jayphbee : `ApeBond` will deploy to multichains including polygon, bsc, arbitrum and ethereum. There is discrepancy regarding the implmenation of `approve`. It will revert if the previous allowance is not fully comsumed on ethereum.

```
    function approve(address _spender, uint _value) public onlyPayloadSize(2 * 32) {

        // To change the approve amount you first have to reduce the addresses`
        //  allowance to zero by calling `approve(_spender, 0)` if it is not
        //  already 0 to mitigate the race condition described here:
        //  https://github.com/ethereum/EIPs/issues/20#issuecomment-263524729
        require(!((_value != 0) && (allowed[msg.sender][_spender] != 0)));

        allowed[msg.sender][_spender] = _value;
        Approval(msg.sender, _spender, _value);
    }
```

This discrepancy could lead to revert when `addLiquidity` function doesn't consume all allowance approved to the `lpRounter` for USDT due to the slippage protection.

```
(vars.amount0Lp, vars.amount1Lp, ) = IUniswapV2Router02(zapParams.liquidityPath.lpRouter).addLiquidity(
                zapParams.token0,
                zapParams.token1,
                vars.amount0Out,
                vars.amount1Out,
                zapParams.liquidityPath.minAmountLP0,
                zapParams.liquidityPath.minAmountLP1,
                zapParams.to,
                zapParams.deadline
            );
```

The impact is that `SoulZap_Univ2#zap` function will always revert on ethereum if the USDT related pair is involved and the allowance to USDT is not fully consumed by the `lpRouter` previously.

Recommendation

Yaodao : Recommend adding logic to reset the allowance to 0 and then calling the new `approve()`, like the `forceApprove()` function in the `OpenZeppelin.SafeERC20`.

Reference: https://github.com/OpenZeppelin/openzeppelin-contracts/blob/master/contracts/token/ERC20/utils/SafeERC20.sol#L76-L83

SerSomeone : use `swapExactTokensForTokensSupportingFeeOnTransferTokens` instead on `swapExactTo-`

`kensForTokens`

parth_15 : Recommend to use `safeApprove` instead and set the allowance to `0` before calling it.

```
function approveUnderlying(address spender) private {
        for (uint256 i = 0; i < weights.length; i++) {
            IERC20(tokens[i]).safeApprove(spender, 0);
            IERC20(tokens[i]).safeApprove(spender, type(uint256).max);
        }
    }
```

Kong7ych3 : It is recommended to perform the approve 0 allowance operation before performing the approve operation to avoid this risk.

jayphbee : Set allowance to zero after `addLiquidity` if the the allowance approved to `lpRouter` is not fully consumed for USDT on ethereum.

Client Response

Fixed. https://github.com/SoulSolidity/SoulZapV1/commit/6c5e05e34074cdcde2ac0c7508960768f92a6877 Changed approve to forceApprove everywhere

APB-3:Taking fee from user Incorrectly in `zapBond()`

Category	Severity	Client Response	Contributor
Logical	Critical	Fixed	Yaodao

Code Reference

- code/contracts/extensions/ApeBond/SoulZap_Ext_ApeBond.sol#L60-L127
- code/contracts/SoulZap_UniV2.sol#L169-L201

```
60:function zapBond(
61:        ZapParams memory zapParams,
62:        SwapPath memory feeSwapPath,
63:        ICustomBillRefillable bond,
64:        uint256 maxPrice
65:    ) external payable nonReentrant whenNotPaused verifyMsgValueAndWrap(zapParams.tokenIn, zapPar
ams.amountIn) {
66:        if (address(zapParams.tokenIn) == address(Constants.NATIVE_ADDRESS)) {
67:            _zapBond(zapParams, feeSwapPath, bond, maxPrice);
68:        } else {
69:            uint256 balanceBefore = _getBalance(zapParams.tokenIn);
70:            zapParams.tokenIn.safeTransferFrom(msg.sender, address(this), zapParams.amountIn);
71:            zapParams.amountIn = _getBalance(zapParams.tokenIn) - balanceBefore;
72:            _zapBond(zapParams, feeSwapPath, bond, maxPrice);
73:        }
74:    }
75:
76:    /// ------------------------------------------------------------------------
77:    /// Private Functions
78:    /// ------------------------------------------------------------------------
79:
80:    function _zapBond(
81:        ZapParams memory zapParams,
82:        SwapPath memory feeSwapPath,
83:        ICustomBillRefillable bond,
84:        uint256 maxPrice
85:    ) private {
86:        IUniswapV2Pair bondPrincipalToken = IUniswapV2Pair(bond.principalToken());
87:        bool skipFee = isBondNftWhitelisted(bond);
88:
89:        //Check if bond principal token is single token or lp
90:        bool isSingleTokenBond = true;
91:        try IUniswapV2Pair(bondPrincipalToken).token0() returns (address /*_token0*/) {
92:            isSingleTokenBond = false;
93:        } catch (bytes memory) {}
94:
95:        address to;
96:        if (isSingleTokenBond) {
97:            SwapParams memory swapParams = SwapParams({
98:                tokenIn: zapParams.tokenIn,
```

```
99:                amountIn: zapParams.amountIn,
100:                tokenOut: zapParams.token0,
101:                path: zapParams.path0,
102:                to: zapParams.to,
103:                deadline: zapParams.deadline
104:            });
105:            require(swapParams.tokenOut == address(bondPrincipalToken), "ApeBond: Wrong token fo
r Bond");
106:            to = swapParams.to;
107:            swapParams.to = address(this);
108:            _swap(swapParams, feeSwapPath, skipFee);
109:        } else {
110:            require(
111:                (zapParams.token0 == bondPrincipalToken.token0() && zapParams.token1 == bondPrin
cipalToken.token1()) ||
112:                    (zapParams.token1 == bondPrincipalToken.token0() &&
113:                        zapParams.token0 == bondPrincipalToken.token1()),
114:                "ApeBond: Wrong LP bondPrincipalToken for Bond"
115:            );
116:            to = zapParams.to;
117:            zapParams.to = address(this);
118:            _zap(zapParams, feeSwapPath, skipFee);
119:        }
120:
121:        uint256 balance = bondPrincipalToken.balanceOf(address(this));
122:        bondPrincipalToken.approve(address(bond), balance);
123:        bond.deposit(balance, maxPrice, to);
124:        bondPrincipalToken.approve(address(bond), 0);
125:
126:        emit ZapBond(zapParams, bond, maxPrice);
127:    }

169:function _swap(SwapParams memory swapParams, SwapPath memory feeSwapPath, bool takeFee) internal
whenNotPaused {
170:        // Verify inputs
171:        require(swapParams.amountIn > 0, "SoulZap: amountIn must be > 0");
172:        require(swapParams.to != address(0), "SoulZap: Can't swap to null address");
173:        require(swapParams.tokenOut != address(0), "SoulZap: tokenOut can't be address(0)");
174:        require(address(swapParams.tokenIn) != swapParams.tokenOut, "SoulZap: tokens can't be th
e same");
175:
176:        bool native = address(swapParams.tokenIn) == address(Constants.NATIVE_ADDRES-
```

```
S);
177:         if (native) swapParams.tokenIn = WNATIVE;
178:
179:         if (takeFee) {
180:             swapParams.amountIn -= _handleFee(
181:                 swapParams.tokenIn,
182:                 swapParams.amountIn,
183:                 feeSwapPath,
184:                 swapParams.deadline
185:             );
186:         }
187:
188:         /**
189:          * Handle token Swap
190:          */
191:         require(swapParams.path.swapRouter != address(0), "SoulZap: swap router can not be addre
ss(0)");
192:         require(swapParams.path.path[0] == address(swapParams.tokenIn), "SoulZap: wrong path pat
h[0]");
193:         require(
194:             swapParams.path.path[swapParams.path.path.length - 1] == swapParams.tokenOut,
195:             "SoulZap: wrong path path[-1]"
196:         );
197:         swapParams.tokenIn.approve(swapParams.path.swapRouter, swapParams.amountIn);
198:         _routerSwapFromPath(swapParams.path, swapParams.amountIn, swapParams.to, swapParams.dead
line);
199:
200:         emit Swap(swapParams);
201:     }
```

Description

Yaodao : The function `zapBond()` will call the internal function `_zapBond()`, and the function `_zapBond()` will call the function `_swap()`. The third parameter of the function `_swap()` is takeFee, which is the standard for whether or not to charge fees.

```
    function _swap(SwapParams memory swapParams, SwapPath memory feeSwapPath, bool takeFee) internal
whenNotPaused {
        // Verify inputs
        require(swapParams.amountIn > 0, "SoulZap: amountIn must be > 0");
        require(swapParams.to != address(0), "SoulZap: Can't swap to null address");
        require(swapParams.tokenOut != address(0), "SoulZap: tokenOut can't be address(0)");
        require(address(swapParams.tokenIn) != swapParams.tokenOut, "SoulZap: tokens can't be the sa
me");

        bool native = address(swapParams.tokenIn) == address(Constants.NATIVE_ADDRESS);
        if (native) swapParams.tokenIn = WNATIVE;

        if (takeFee) {
            swapParams.amountIn -= _handleFee(
                swapParams.tokenIn,
                swapParams.amountIn,
                feeSwapPath,
                swapParams.deadline
            );
        }

        /**
         * Handle token Swap
         */
        require(swapParams.path.swapRouter != address(0), "SoulZap: swap router can not be address
(0)");
        require(swapParams.path.path[0] == address(swapParams.tokenIn), "SoulZap: wrong path path
[0]");
        require(
            swapParams.path.path[swapParams.path.path.length - 1] == swapParams.tokenOut,
            "SoulZap: wrong path path[-1]"
        );
        swapParams.tokenIn.approve(swapParams.path.swapRouter, swapParams.amountIn); // @audit unche
ck return value
        _routerSwapFromPath(swapParams.path, swapParams.amountIn, swapParams.to, swapParams.deadlin
e);

        emit Swap(swapParams);
    }
```

However, in the function `_zapBond()`, the result of the function `isBondNftWhitelisted()` is used as the parameter `takeFee` to call the function `_swap()`. As a result, the bond in whitelist will be charged fees, and the bond not in whitelist will not be charged fees.

```
    function _zapBond(
        ZapParams memory zapParams,
        SwapPath memory feeSwapPath,
        ICustomBillRefillable bond,
        uint256 maxPrice
    ) private {
        IUniswapV2Pair bondPrincipalToken = IUniswapV2Pair(bond.principalToken());
        bool skipFee = isBondNftWhitelisted(bond);

        //Check if bond principal token is single token or lp
        bool isSingleTokenBond = true;
        try IUniswapV2Pair(bondPrincipalToken).token0() returns (address /*_token0*/) {
            isSingleTokenBond = false;
        } catch (bytes memory) {}

        address to;
        if (isSingleTokenBond) {
            SwapParams memory swapParams = SwapParams({
                tokenIn: zapParams.tokenIn,
                amountIn: zapParams.amountIn,
                tokenOut: zapParams.token0,
                path: zapParams.path0,
                to: zapParams.to,
                deadline: zapParams.deadline
            });
            require(swapParams.tokenOut == address(bondPrincipalToken), "ApeBond: Wrong token for Bo
nd");
            to = swapParams.to;
            swapParams.to = address(this);
            _swap(swapParams, feeSwapPath, skipFee);
        } else {
            require(
                (zapParams.token0 == bondPrincipalToken.token0() && zapParams.token1 == bondPrincipa
lToken.token1()) ||
                    (zapParams.token1 == bondPrincipalToken.token0() &&
                        zapParams.token0 == bondPrincipalToken.token1()),
                "ApeBond: Wrong LP bondPrincipalToken for Bond"
            );
            to = zapParams.to;
            zapParams.to = address(this);
            _zap(zapParams, feeSwapPath, skipFee);
        }
```

Recommendation

Yaodao : Recommend using `!skipFee` as the parameter `takeFee` to call the function `_swap()`

Client Response

Fixed. https://github.com/SoulSolidity/SoulZapV1/commit/dc1bf836d17f0374c82bfe49dd9415a2efdec158

APB-4: `SoulZap_UniV2._zap()` function has approval to router contract even after completion of transaction

Category	Severity	Client Response	Contributor
Privilege Related	Critical	Fixed	parth_15

Code Reference

- code/contracts/SoulZap_UniV2.sol#L312-L313
- code/contracts/SoulZap_UniV2.sol#L331-L338

```
312:IERC20(zapParams.token0).approve(address(zapParams.liquidityPath.lpRouter), vars.amount0Out);
313:        IERC20(zapParams.token1).approve(address(zapParams.liquidityPath.lpRouter), vars.amount1
Out);

331:if (zapParams.token0 == address(WNATIVE)) {
332:            // Ensure WNATIVE is called last
333:            _transferOut(IERC20(zapParams.token1), vars.amount1Out - vars.amount1Lp, msg.sender,
native);
334:            _transferOut(IERC20(zapParams.token0), vars.amount0Out - vars.amount0Lp, msg.sender,
native);
335:        } else {
336:            _transferOut(IERC20(zapParams.token0), vars.amount0Out - vars.amount0Lp, msg.sender,
native);
337:            _transferOut(IERC20(zapParams.token1), vars.amount1Out - vars.amount1Lp, msg.sender,
native);
338:        }
```

Description

parth_15 : The `_zap` function gives approval to `zapParams.liquidityPath.lpRouter`. This approval is needed because `zapParams.liquidityPath.lpRouter` will fetch the funds from the `SoulZap_UniV2` while zapping(providing liquidity) to uniswap v2. The approval is given as follows:

```
IERC20(zapParams.token0).approve(address(zapParams.liquidityPath.lpRouter), vars.amount0Out);
IERC20(zapParams.token1).approve(address(zapParams.liquidityPath.lpRouter), vars.amount1Out);
```

As we can notice, the approval of `vars.amount0Out` is given for `zapParams.token0` token and `vars.amount1Out` is given for `zapParams.token1` token. Now, `zapParams.liquidityPath.lpRouter` can fetch the amount

approved. But it is not guaranteed that `zapParams.liquidityPath.lpRouter` will fetch the exact approved amount. That is why the contract is refunding the amount that was not sent to `zapParams.liquidityPath.lpRouter` here. But even after transaction is complete, the approval to the remaining amount which was not fetched by `zapParams.liquidityPath.lpRouter` and is refunded to the user is still there to `zapParams.liquidityPath.lpRouter`. Also, other point to note that `zapParams.liquidityPath.lpRouter` is user controlled and user controlled router can gain approval from `SoulZap_UniV2`.

Recommendation

parth_15 : The above issue can be mitigated by resetting the token approvals to `0` at the end of the transaction to ensure that the `SoulZap_UniV2` doesn't have any pending approvals to uniswap v2 router or any other user specificed router.

Client Response

Fixed,https://github.com/SoulSolidity/SoulZapV1/commit/9f567543546e33893746e635e12805b87af13169 Fixed issue as recommended

APB-5:Underflow error in function `SoulFeeManager.getFee()`

Category	Severity	Client Response	Contributor
Integer Overflow and Underflow	Critical	Fixed	parth_15, TrungOre

Code Reference

- code/contracts/fee-manager/SoulFeeManager.sol#L155-L161
- code/contracts/fee-manager/SoulFeeManager.sol#L155-L162

```
155:function getFee(uint256 epochFeeVolume) public view returns (uint256 fee) {
156:        for (uint256 i = volumeFeeThresholds.length - 1; i >= 0; i--) {
157:            if (epochFeeVolume >= volumeFeeThresholds[i].volume) {
158:                return volumeFeeThresholds[i].fee;
159:            }
160:        }
161:        return 0;

155:function getFee(uint256 epochFeeVolume) public view returns (uint256 fee) {
156:        for (uint256 i = volumeFeeThresholds.length - 1; i >= 0; i--) {
157:            if (epochFeeVolume >= volumeFeeThresholds[i].volume) {
158:                return volumeFeeThresholds[i].fee;
159:            }
160:        }
161:        return 0;
162:    }
```

Description

parth_15 : The `getFee` function is as follows:

```
function getFee(uint256 epochFeeVolume) public view returns (uint256 fee) {
        for (uint256 i = volumeFeeThresholds.length - 1; i >= 0; i--) {
            if (epochFeeVolume >= volumeFeeThresholds[i].volume) {
                return volumeFeeThresholds[i].fee;
            }
        }
        return 0;
    }
```

It retrieves the fee based on the provided epoch fee volume. The above for-loop can revert due to underflow because when `i` is `0`, `i--` will underflow. So, due to this `swap` and `zap` functionality will never work when protocol is launched because `epochFeeVolume` will always be `0` at the start. Due to not working of this function, there won't be any possible `swap` or `zap` and no fee will be accumulated ever. So, protocol state won't change and it won't work at any time.

TrungOre : The `SoulFeeManager.getFee()` function is employed to retrieve the fee based on the provided epoch fee volume. This function iterates from `volumeFeeThresholds.length - 1` down to `0` to identify the index `i` satisfying the condition `epochFeeVolume >= volumeFeeThresholds[i].volume`. It then returns the corresponding `volumeFeeThresholds[i].fee`. In cases where no index is found, the function considers the fee percentage for that volume as `0`.

However, there is a potential underflow issue in the for-loop. Notably, the value of the iteration variable `i` can become `0`. When this occurs due to the `i--` operation, the value of `i` in the last iteration becomes `0 - 1 = -1`, resulting in an underflow error.

Impact: The `SoulFeeManager.getFee()` function is integral to the `SoulFeeManager.getFeeInfo()` function, which, in turn, is utilized by the `SoulZap_UniV2._handleFee()` function to calculate the fee input. Due to the underflow issue, this function would malfunction, leading to the failure of the core functions `swap()` / `zap()`, resulting in a revert.

Recommendation

parth_15 : Change the logic of above for-loop as below:

```
function getFee(uint256 epochFeeVolume) public view returns (uint256 fee) {
        for (uint256 i = volumeFeeThresholds.length; i >= 1; i--) {
            if (epochFeeVolume >= volumeFeeThresholds[i-1].volume) {
                return volumeFeeThresholds[i-1].fee;
            }
        }
        return 0;
    }
```

TrungOre : Consider modifying the function `SoulFeeManager.getFee()` as follows:

```
function getFee(uint256 epochFeeVolume) public view returns (uint256 fee) {
    for (uint256 i = volumeFeeThresholds.length; i > 0; i--) {
        if (epochFeeVolume >= volumeFeeThresholds[i-1].volume) {
            return volumeFeeThresholds[i-1].fee;
        }
    }
    return 0;
}
```

Client Response

Fixed,https://github.com/SoulSolidity/SoulZapV1/commit/9bf3d1b9fb5097dfe6fa97621e272ba4ac2bfb63

```
    function getFee(uint256 epochFeeVolume) public view returns (uint256 fee) {
        for (uint256 i = volumeFeeThresholds.length; i > 0; i--) {
            if (epochFeeVolume >= volumeFeeThresholds[i - 1].volume) {
                return volumeFeeThresholds[i - 1].fee;
            }
        }
        return 0;
    }
```

APB-6:Missing check `msg.value == 0` in `SoulZap::verifyMsgValueAndWrap`

Category	Severity	Client Response	Contributor
Logical	Critical	Fixed	infinityhacker

Code Reference

- code/contracts/SoulZap_UniV2.sol#L111-L114

```
111:modifier verifyMsgValueAndWrap(IERC20 _inputToken, uint256 _inputAmount) {
112:         if (msg.value > 0) {
113:             require(
114:                 address(_inputToken) == address(Constants.NATIVE_ADDRESS),
```

Description

infinityhacker : In `swap` function, there is a modifier call `verifyMsgValueAndWrap`, which accepted two parameters from `swapParams`, to determine how much native token user paid and help user to convert it to wrap tokens, but according to the logic in `verifyMsgValueAndWrap`,

```
modifier verifyMsgValueAndWrap(IERC20 _inputToken, uint256 _inputAmount) {
        if (msg.value > 0) {
            require(
                address(_inputToken) == address(Constants.NATIVE_ADDRESS),
                "SoulZap: tokenIn MUST be NATIVE_ADDRESS with msg.value"
            );
            (, uint256 wrappedAmount) = _wrapNative();
            require(_inputAmount == wrappedAmount, "SoulZap: amountIn not equal to wrappedAmount");
        } // missing check on msg.value == 0
        _;
    }
```

there is a flaw, that is, the modifier missing the check on `msg.value == 0`, if `msg.value ==0`, the check `require(_inputAmount == wrappedAmount, "SoulZap: amountIn not equal to wrappedAmount");` will not work, and next, the `_inputAmount` from `swapParams` will not be check and the contract will think that user has deposited `_inputAmount` of native tokens.

Recommendation

infinityhacker : Add `msg.value == 0` check on `verifyMsgValueAndWrap` modifier

Client Response

Fixed,https://github.com/SoulSolidity/SoulZapV1/commit/e08b55d1706ce7f672eb36734515570651d1ec3d

```
    modifier verifyMsgValueAndWrap(IERC20 _inputToken, uint256 _inputAmount) {
        if (address(_inputToken) == address(Constants.NATIVE_ADDRESS)) {
            (, uint256 wrappedAmount) = _wrapNative();
            require(_inputAmount == wrappedAmount, "SoulZap: amountIn not equal to wrappedAmount");
        } else {
            require(msg.value == 0, "SoulZap: msg.value should be 0");
        }
        _;
    }
```

APB-7:Lower bound fee rate for `SoulFeeManager::getFee()` is incorrect

Category	Severity	Client Response	Contributor
Logical	Medium	Fixed	ravikiran_web3

Code Reference

- code/contracts/fee-manager/SoulFeeManager.sol#L155-L162

```
155:function getFee(uint256 epochFeeVolume) public view returns (uint256 fee) {
156:        for (uint256 i = volumeFeeThresholds.length - 1; i >= 0; i--) {
157:            if (epochFeeVolume >= volumeFeeThresholds[i].volume) {
158:                return volumeFeeThresholds[i].fee;
159:            }
160:        }
161:        return 0;
162:    }
```

Description

ravikiran_web3 : The implementation logic does not consider the lower bound scenario.

Example, lets say, the configured threshold values are as below.

[{ volume: 5000, fee:5 },{ volume: 6000, fee:6 },{ volume: 7000, fee:7 },{ volume: 8000, fee:8 },{ volume: 9000, fee:9 }]

a) check for input Volume = 9200 Since Input Volume is greater than 9000, fee applicable is 9

b) check for input volume = 8999 Since the input volume is greater than 8000, fee applicable is 8

c) check for input volume = 4999 Since the input volume is not greater than any configured volume, the fee applicable is 0

This gap could arise due to human error during configuration.

Recommendation

ravikiran_web3 : The suggestion is to apply the fee for the lowest index incase the volume is lower than the least configured volume threshold.

Suggested fix is as below where instead of returning 0, return the lowest configured fee.

```
function getFee(uint256 epochFeeVolume) public view returns (uint256 fee) {
for (uint256 i = volumeFeeThresholds.length - 1; i >= 0; i--) {
  if (epochFeeVolume >= volumeFeeThresholds[i].volume) {
   return volumeFeeThresholds[i].fee;
  }
}
   return volumeFeeThresholds[0].fee;
}
```

Client Response

Fixed. https://github.com/SoulSolidity/SoulZapV1/commit/2ef4c50a9e3fef063ff7d9e64cb1cc803e26ea8a Fixed as recommended

APB-8:ArrakisMath inconsistencies for different tokens decimals

Category	Severity	Client Response	Contributor
Integer Overflow and Underflow	**Medium**	**Fixed**	**ginlee, ravikiran_web3, BradMoonUESTC, jayphbee**

Code Reference

- code/contracts/extensions/Arrakis/lib/ArrakisMath.sol#L50-L92
- code/contracts/extensions/Arrakis/lib/ArrakisMath.sol#L59-L60
- code/contracts/extensions/Arrakis/lib/ArrakisMath.sol#L97-L99
- code/contracts/extensions/Arrakis/lib/ArrakisMath.sol#L97
- code/contracts/extensions/Arrakis/lib/ArrakisMath.sol#L156-L160
- code/contracts/SoulZap_UniV2.sol#L444
- code/contracts/SoulZap_UniV2.sol#L451

```
50:function getSwapRatio(
51:        SwapRatioParams memory swapRatioParams
52:    ) internal view returns (uint256 amount0, uint256 amount1) {
53:        SwapRatioLocalVars memory vars;
54:
55:        (vars.underlying0, vars.underlying1) = IArrakisPool(swapRatioParams.arrakisPool).getUnder
lyingBalances();
56:
57:        vars.token0decimals = ERC20(address(swapRatioParams.token0)).decimals();
58:        vars.token1decimals = ERC20(address(swapRatioParams.token1)).decimals();
59:        vars.underlying0 = _normalizeTokenDecimals(vars.underlying0, vars.token0decimals);
60:        vars.underlying1 = _normalizeTokenDecimals(vars.underlying1, vars.token1decimals);
61:
62:        vars.weightedPrice0 = swapRatioParams.inputToken == swapRatioParams.token0
63:            ? 1e18
64:            : getWeightedPrice(
65:                swapRatioParams.path0,
66:                swapRatioParams.uniV3PoolFees0,
67:                swapRatioParams.uniV2Router0,
68:                swapRatioParams.uniV3Factory
69:            );
70:        vars.weightedPrice1 = swapRatioParams.inputToken == swapRatioParams.token1
71:            ? 1e18
72:            : getWeightedPrice(
73:                swapRatioParams.path1,
74:                swapRatioParams.uniV3PoolFees1,
75:                swapRatioParams.uniV2Router1,
76:                swapRatioParams.uniV3Factory
77:            );
78:
79:        vars.percentage0 =
80:            (((vars.underlying0 * 1e18) / (vars.underlying0 + vars.underlying1)) * vars.weightedP
rice0) /
81:            (vars.weightedPrice0 + vars.weightedPrice1);
82:
83:        vars.percentage1 =
84:            (((vars.underlying1 * 1e18) / (vars.underlying0 + vars.underlying1)) * vars.weightedP
rice1) /
85:            (vars.weightedPrice0 + vars.weightedPrice1);
86:
```

```
87:         amount0 =
88:             (((vars.percentage0 * 1e18) / (vars.percentage0 + vars.percentage1)) * swapRatioParam
s.inputAmount) /
89:             1e18;
90:
91:         amount1 = swapRatioParams.inputAmount - amount0;
92:     }

59:vars.underlying0 = _normalizeTokenDecimals(vars.underlying0, vars.token0decimals);
60:         vars.underlying1 = _normalizeTokenDecimals(vars.underlying1, vars.token1decimals);

97:function _normalizeTokenDecimals(uint256 amount, uint256 decimals) internal pure returns (uint25
6) {
98:         return amount * 10 ** (18 - decimals);
99:     }

97:function _normalizeTokenDecimals(uint256 amount, uint256 decimals) internal pure returns (uint25
6) {
98:         return amount * 10 ** (18 - decimals);
99:     }

97:function _normalizeTokenDecimals(uint256 amount, uint256 decimals) internal pure returns (uint25
6) {

156:if (token1 < token0) {
157:             price = (2 ** 192) / ((sqrtPriceX96) ** 2 / uint256(10 ** (token0Decimals + 18 - tok
en1Decimals)));
158:         } else {
159:             price = ((sqrtPriceX96) ** 2) / ((2 ** 192) / uint256(10 ** (token0Decimals + 18 - t
oken1Decimals)));
160:         }

444:_accumulateFeeVolume(amountOut);

451:_accumulateFeeVolume(inputFeeAmount);
```

Description

ginlee :

```
 vars.underlying0 = _normalizeTokenDecimals(vars.underlying0, vars.token0decimals);
 vars.underlying1 = _normalizeTokenDecimals(vars.underlying1, vars.token1decimals);

function _normalizeTokenDecimals(uint256 amount, uint256 decimals) internal pure returns (uint256) {
        return amount * 10 ** (18 - decimals);
    }
```

function _normalizeTokenDecimals will revert when underlying token has higher than 18 decimals, even though in function _normalizeTokenDecimals comment it is said

```
/// @param decimals Decimals of given token amount to scale. MUST be <=18
```

but when using this function, there is no guarantee that token0decimals or token1decimals can't be more than 18, This will cause inabilities for calculating vars.underlying0 and vars.underlying1

ravikiran_web3 : The library code assumes that token will have a max of 18 decimal positions which can cause the integer overflow. But, there are tokens that have more than 18 decimal positions.

```
 function _normalizeTokenDecimals(uint256 amount, uint256 decimals) internal pure returns (uint256)
{
        return amount * 10 ** (18 - decimals);
    }
```

and

```
if (token1 < token0) {
            price = (2 ** 192) / ((sqrtPriceX96) ** 2 / uint256(10 ** (token0Decimals + 18 - token1D
ecimals)));
        } else {
            price = ((sqrtPriceX96) ** 2) / ((2 ** 192) / uint256(10 ** (token0Decimals + 18 - token
1Decimals)));
        }
```

BradMoonUESTC : The smart contract code provided exhibits potential vulnerabilities in the handling of decimal precision and price calculation within the `getSwapRatio` and related functions. Two primary issues are identified:

1. **Inconsistent Decimal Normalization**: The `_normalizeTokenDecimals` function normalizes token amounts to 18 decimal places, a standard practice in Ethereum contracts. However, this normalization might not consistently align with the precision of price ratios obtained from oracles or the `getWeightedPrice` function. Any mismatch in decimal handling can result in inaccurate calculations of swap ratios and percentages, crucial for financial transactions.
2. **Incorrect Price Calculation with Hardcoded Constants**: The code uses a hardcoded normalization factor `(2 ** 192)` for calculating the price based on the square of `sqrtPriceX96`. This approach assumes a uniform structure for price values and might not be applicable to all token pairs, especially when `token1Decimals` is greater than `token0Decimals`. Such scenarios can lead to inflated or inaccurate price outputs, potentially causing financial discrepancies.

   jayphbee : User's fee is accumulated in `_handleFee` function by calling `_accumulateFeeVolume` internally.

```
function _handleFee(
        IERC20 _inputToken,
        uint256 _inputAmount,
        SwapPath memory _feeSwapPath,
        uint256 _deadline
    ) private returns (uint256 inputFeeAmount) {
        ...
        inputFeeAmount = (_inputAmount * feePercentage) / feeDenominator;

        if (_feeSwapPath.path.length >= 2) {
            ...
            uint256 amountOut = _routerSwapFromPath(_feeSwapPath, inputFeeAmount, feeCollector, _dea
dline);
            _accumulateFeeVolume(amountOut);
        } else {
            /// @dev Input token is considered fee token or a token with no output route
            /// In order to not create a denial of service, we take any input token in this case.
            _transferOut(_inputToken, inputFeeAmount, feeCollector, false);
            // Only increase fee volume if input token is a fee token
            if (soulFeeManager.isFeeToken(address(_inputToken))) {
                _accumulateFeeVolume(inputFeeAmount);
            }
        }
    }
```

And the fee amount is determined by `epochFeeVolume`

```
    function getFee(uint256 epochFeeVolume) public view returns (uint256 fee) {
        for (uint256 i = volumeFeeThresholds.length - 1; i >= 0; i--) {
            if (epochFeeVolume >= volumeFeeThresholds[i].volume) {
                return volumeFeeThresholds[i].fee;
            }
        }
        return 0;
    }
```

We can see that `amountOut` and `inputFeeAmount` is directly accumulated without normalization. Let's say `USDC` and `DAI` are fee tokens, 1 USDC should have the similar value with 1 DAI, but DAI will contribute `10^12` times large fee volume than USDC because USDC has decimals 6 and DAI has decimals 18.

The impact is that the protocol has the advantage to use some large decimal tokens to accumulate fee volume rapidly thus users have to suffer higher fee tier.

Recommendation

ginlee : Consider modifying how _normalizeTokenDecimals works so it could handle tokens with higher than 18 decimals.
ravikiran_web3 : At the entry point of each applicable library function, validate if the token has decimal positions to be less than or equal to 18 decimals. For any tokens have larger than 18 decimals reverts with error stating the token is not supported.
BradMoonUESTC : To address these vulnerabilities, the following steps are recommended:

- **Review and Standardize Decimal Handling**: Ensure consistent handling of decimals throughout the contract, particularly in functions involving financial calculations. This includes revising the normalization process in `getWeighTedPrice` and related functions to accurately reflect the decimal precision of the tokens and price feeds.
- **Dynamic Normalization Factors**: Replace hardcoded constants with dynamic calculations that consider the actual decimals of the tokens involved. This will help in accurately scaling the price ratios and avoiding inflated outputs.
- **Comprehensive Testing and Auditing**: Conduct thorough testing, including edge cases where token decimals significantly differ. Additionally, consider a third-party audit to identify and rectify any overlooked issues, ensuring the contract's accuracy and security in handling financial transactions.

Implementing these recommendations will enhance the reliability and precision of the smart contract, especially in critical financial operations like token swap ratios.
jayphbee : Normalize fee amount to a unified token like USDT or ETH.

Client Response

Fixed

APB-9:Inaccurate Token Amount Calculation in Liquidity Addition

Category	Severity	Client Response	Contributor
Logical	Medium	Acknowledged	parth_15, BradMoonUESTC

Code Reference

- code/contracts/SoulZap_UniV2.sol#L237-L341
- code/contracts/SoulZap_UniV2.sol#L272-L273
- code/contracts/SoulZap_UniV2.sol#L281-L307

```
237:function _zap(ZapParams memory zapParams, SwapPath memory feeSwapPath, bool takeFee) internal wh
enNotPaused {
238:        // Verify inputs
239:        require(zapParams.amountIn > 0, "SoulZap: amountIn must be > 0");
240:        require(zapParams.to != address(0), "SoulZap: Can't zap to null address");
241:        require(zapParams.liquidityPath.lpRouter != address(0), "SoulZap: lp router can not be a
ddress(0)");
242:        require(zapParams.token0 != address(0), "SoulZap: token0 can not be address(0)");
243:        require(zapParams.token1 != address(0), "SoulZap: token1 can not be address(0)");
244:
245:        bool native = address(zapParams.tokenIn) == address(Constants.NATIVE_ADDRESS);
246:        if (native) zapParams.tokenIn = WNATIVE;
247:
248:        // Setup struct to prevent stack overflow
249:        LocalVarsLib.LocalVars memory vars;
250:        // Ensure token addresses and paths are in ascending numerical order
251:        if (zapParams.token1 < zapParams.token0) {
252:            (zapParams.token0, zapParams.token1) = (zapParams.token1, zapParams.token0);
253:            (zapParams.path0, zapParams.path1) = (zapParams.path1, zapParams.path0);
254:        }
255:
256:        if (takeFee) {
257:            zapParams.amountIn -= _handleFee(zapParams.tokenIn, zapParams.amountIn, feeSwapPath,
zapParams.deadline);
258:        }
259:
260:        /**
261:         * Setup swap amount0 and amount1
262:         */
263:        if (zapParams.liquidityPath.lpType == LPType.V2) {
264:            // Handle UniswapV2 Liquidity
265:            require(
266:                IUniswapV2Factory(IUniswapV2Router02(zapParams.liquidityPath.lpRouter).factory
()).getPair(
267:                    zapParams.token0,
268:                    zapParams.token1
269:                ) != address(0),
270:                "SoulZap: Pair doesn't exist"
```

```
271:             );
272:             vars.amount0In = zapParams.amountIn / 2;
273:             vars.amount1In = zapParams.amountIn / 2;
274:         } else {
275:             revert("SoulZap: LPType not supported");
276:         }
277:
278:         /**
279:          * Handle token0 Swap
280:          */
281:         if (zapParams.token0 != address(zapParams.tokenIn)) {
282:             require(zapParams.path0.swapRouter != address(0), "SoulZap: swap router can not be a
ddress(0)");
283:             require(zapParams.path0.path[0] == address(zapParams.tokenIn), "SoulZap: wrong path
path0[0]");
284:             require(
285:                 zapParams.path0.path[zapParams.path0.path.length - 1] == zapParams.token0,
286:                 "SoulZap: wrong path path0[-1]"
287:             );
288:             zapParams.tokenIn.approve(zapParams.path0.swapRouter, vars.amount0In);
289:             vars.amount0Out = _routerSwapFromPath(zapParams.path0, vars.amount0In, address(thi
s), zapParams.deadline);
290:         } else {
291:             vars.amount0Out = zapParams.amountIn - vars.amount1In;
292:         }
293:         /**
294:          * Handle token1 Swap
295:          */
296:         if (zapParams.token1 != address(zapParams.tokenIn)) {
297:             require(zapParams.path1.swapRouter != address(0), "SoulZap: swap router can not be a
ddress(0)");
298:             require(zapParams.path1.path[0] == address(zapParams.tokenIn), "SoulZap: wrong path
path1[0]");
299:             require(
300:                 zapParams.path1.path[zapParams.path1.path.length - 1] == zapParams.token1,
301:                 "SoulZap: wrong path path1[-1]"
302:             );
303:             zapParams.tokenIn.approve(zapParams.path1.swapRouter, vars.amount1In);
304:             vars.amount1Out = _routerSwapFromPath(zapParams.path1, vars.amount1In, address(thi
s), zapParams.deadline);
305:         } else {
```

```
306:             vars.amount1Out = zapParams.amountIn - vars.amount0In;
307:         }
308:
309:         /**
310:          * Handle Liquidity Add
311:          */
312:         IERC20(zapParams.token0).approve(address(zapParams.liquidityPath.lpRouter), vars.amount0
Out);
313:         IERC20(zapParams.token1).approve(address(zapParams.liquidityPath.lpRouter), vars.amount1
Out);
314:
315:         if (zapParams.liquidityPath.lpType == LPType.V2) {
316:             // Add liquidity to UniswapV2 Pool
317:             (vars.amount0Lp, vars.amount1Lp, ) = IUniswapV2Router02(zapParams.liquidityPath.lpRo
uter).addLiquidity(
318:                 zapParams.token0,
319:                 zapParams.token1,
320:                 vars.amount0Out,
321:                 vars.amount1Out,
322:                 zapParams.liquidityPath.minAmountLP0,
323:                 zapParams.liquidityPath.minAmountLP1,
324:                 zapParams.to,
325:                 zapParams.deadline
326:             );
327:         } else {
328:             revert("SoulZap: lpType not supported");
329:         }
330:
331:         if (zapParams.token0 == address(WNATIVE)) {
332:             // Ensure WNATIVE is called last
333:             _transferOut(IERC20(zapParams.token1), vars.amount1Out - vars.amount1Lp, msg.sender,
native);
334:             _transferOut(IERC20(zapParams.token0), vars.amount0Out - vars.amount0Lp, msg.sender,
native);
335:         } else {
336:             _transferOut(IERC20(zapParams.token0), vars.amount0Out - vars.amount0Lp, msg.sender,
native);
337:             _transferOut(IERC20(zapParams.token1), vars.amount1Out - vars.amount1Lp, msg.sender,
native);
338:         }
339:
340:         emit Zap(zapParams);
341:     }
```

```
272:vars.amount0In = zapParams.amountIn / 2;
273:            vars.amount1In = zapParams.amountIn / 2;

281:if (zapParams.token0 != address(zapParams.tokenIn)) {
282:            require(zapParams.path0.swapRouter != address(0), "SoulZap: swap router can not be a
ddress(0)");
283:            require(zapParams.path0.path[0] == address(zapParams.tokenIn), "SoulZap: wrong path
path0[0]");
284:            require(
285:                zapParams.path0.path[zapParams.path0.path.length - 1] == zapParams.token0,
286:                "SoulZap: wrong path path0[-1]"
287:            );
288:            zapParams.tokenIn.approve(zapParams.path0.swapRouter, vars.amount0In);
289:            vars.amount0Out = _routerSwapFromPath(zapParams.path0, vars.amount0In, address(thi
s), zapParams.deadline);
290:        } else {
291:            vars.amount0Out = zapParams.amountIn - vars.amount1In;
292:        }
293:        /**
294:         * Handle token1 Swap
295:         */
296:        if (zapParams.token1 != address(zapParams.tokenIn)) {
297:            require(zapParams.path1.swapRouter != address(0), "SoulZap: swap router can not be a
ddress(0)");
298:            require(zapParams.path1.path[0] == address(zapParams.tokenIn), "SoulZap: wrong path
path1[0]");
299:            require(
300:                zapParams.path1.path[zapParams.path1.path.length - 1] == zapParams.token1,
301:                "SoulZap: wrong path path1[-1]"
302:            );
303:            zapParams.tokenIn.approve(zapParams.path1.swapRouter, vars.amount1In);
304:            vars.amount1Out = _routerSwapFromPath(zapParams.path1, vars.amount1In, address(thi
s), zapParams.deadline);
305:        } else {
306:            vars.amount1Out = zapParams.amountIn - vars.amount0In;
307:        }
```

Description

parth_15 : The `_zap` function uses 50:50 split of `amountIn` of `tokenIn` to supply to the uniswap v2 pool. This is highly unefficient strategy. In this strategy, there are also chances of unutilized assets remaining in user's balance.

Two major problems arise when swapping from one asset to another:

`Swap fee` (0.3% for Uniswap) - With swap fee, the user receives slightly less amount of the swap out asset.

`The new reserve's asset ratio` - The swap alters the reserve ratio, increasing the amount of supplied asset and decreasing the amount of the withdrawn asset. Thus, the optimal swap amount needs to take into account these issues.

In this article, they explored formula (with proofs!) to solve the optimal swap amount for supplying one-sided liquidity.

For further context, zapper also used the formula mentioned in the above article for depositing single sided liquidity in uniswap v2 pools.

BradMoonUESTC : The vulnerability identified in the smart contract relates to the imprecise calculation of token amounts when adding liquidity to a pool. Specifically, the issue arises in the following code segment:

```
vars.amount0In = zapParams.amountIn / 2;
vars.amount1In = zapParams.amountIn / 2;
```

Here, the contract divides the input amount (`amountIn`) equally between `amount0In` and `amount1In`, ignoring the actual token reserve ratio required in the liquidity pool. This naive division can lead to an imbalance in the provision of liquidity. The problem manifests when the `addLiquidity` function is called with these imbalanced inputs, potentially resulting in users contributing more of one token than necessary to the liquidity pool, thereby incurring unintended losses.

Recommendation

parth_15 : Consider using the formula proved in above article to zap into uniswap v2 pools efficiently.

BradMoonUESTC : To mitigate this vulnerability, the contract should dynamically calculate the token amounts based on the current reserve ratios in the liquidity pool. This can be achieved by querying the pool's current state to determine the optimal ratio of tokens required for liquidity provision. Implementing a function that adjusts the input token amounts to align with the pool's existing reserve ratio will ensure a balanced and efficient addition of liquidity.

Client Response

Acknowledged,This is a great find. Thanks for reporting. Due to our current limitations we won't be providing an update for this in this version. Tracking here: https://github.com/SoulSolidity/SoulZapV1/issues/13

APB-10:Lack of checking actual tokens received (Deflationary token)

Category	Severity	Client Response	Contributor
Logical	Medium	Fixed	Kong7ych3

Code Reference

- code/contracts/SoulZap_UniV2.sol#L374

```
374:IUniswapV2Router02(router).swapExactTokensForTokens(amountIn, amountOutMin, path, _to, deadlin
e);
```

Description

Kong7ych3 : In the SoulZap_UniV2 contract, the `_routerSwap` function is used to call the swapExactTokensForTokens function of the specified router to complete the token swap. But for deflationary tokens, the swapExactTokensForTokens function cannot complete the conversion operation, so deflationary tokens will be charged during the transfer process. This will make the SoulZap_UniV2 contract unable to provide zap and swap services for deflationary tokens.

It should be noted that the GNANA token of the Apebond protocol is also a deflationary token, so the SoulZap_UniV2 contract is not compatible with the protocol's own local token, which must be resolved.

Recommendation

Kong7ych3 : It is recommended to use the try-catch function for token swap, try the swapExactTokensForTokens operation first, and then perform the swapExactTokensForTokensSupportingFeeOnTransferTokens operation after failure. Consider the solution:

```
    function _routerSwap(
        ...
    ) private {
        if (swapType == SwapType.V2) {
          try
            IUniswapV2Router02(router).swapExactTokensForTokens(amountIn, amountOutMin, path, _to, d
eadline)
          {} catch Error(string memory /*reason*/) {
            IUniswapV2Router02(router).swapExactTokensForTokensSupportingFeeOnTransferTokens(...);
          } ...
        } else {
            revert("SoulZap: SwapType not supported");
        }
```

Client Response

Fixed,Fixed on this commit/line: https://github.com/SoulSolidity/SoulZapV1/commit/6c5e05e34074cdcde2ac0c7508960768f92a6877#diff-a02d7191256c1597f5e39c65a0a2fe851162c1b00d185de2916fa5346cf42912R381

APB-11:Addressing Duplicate Addresses in Token Swap Paths for Enhanced Security and Efficiency

Category	Severity	Client Response	Contributor
Logical	Medium	Fixed	TrungOre, BradMoonUESTC

Code Reference

- code/contracts/SoulZap_UniV2.sol#L343-L362
- code/contracts/SoulZap_UniV2_Lens.sol#L490-L503

```
343:function _routerSwapFromPath(
344:        SwapPath memory _uniSwapPath,
345:        uint256 _amountIn,
346:        address _to,
347:        uint256 _deadline
348:    ) private returns (uint256 amountOut) {
349:        require(_uniSwapPath.path.length >= 2, "SoulZap: need path0 of >=2");
350:        address outputToken = _uniSwapPath.path[_uniSwapPath.path.length - 1];
351:        uint256 balanceBefore = _getBalance(IERC20(outputToken), _to);
352:        _routerSwap(
353:            _uniSwapPath.swapRouter,
354:            _uniSwapPath.swapType,
355:            _amountIn,
356:            _uniSwapPath.amountOutMin,
357:            _uniSwapPath.path,
358:            _to,
359:            _deadline
360:        );
361:        amountOut = _getBalance(IERC20(outputToken), _to) - balanceBefore;
362:    }

490:// Construct the path through the two hop tokens
491:                address[] memory path = new address[](4);
492:                path[0] = _fromToken;
493:                path[1] = fromTokenHopTokens[i];
494:                path[2] = toTokenHopTokens[j];
495:                path[3] = _toToken;
496:                /// @dev Code duplication in sharedHopTokens section
497:                // Calculate the output amount for this path
498:                uint256 amountOut = calculateOutputAmount(_amountIn, path);
499:
500:                // Update the best path and best amount out min if this path is better
501:                if (amountOut > bestAmountOutMin) {
502:                    bestPath = path;
503:                    bestAmountOutMin = amountOut;
```

Description

TrungOre : The `SoulZap_UniV2_Lens._getBestPath()` function is designed to construct the optimal path for generating the highest possible amount of `_toToken` from a given `_amountIn` of `_fromToken`. Initially, the function identifies all potential hop tokens for swapping between `_fromToken` and `_toToken` by invoking the `findPossible-`

`HopTokens()` function. The results are stored in three separate arrays: `sharedHopTokens`, `fromTokenHopTokens`, and `toTokenHopTokens`.

The function then considers three scenarios corresponding to the length of the swap path (ranging from 2 to 4). In the case where the length of the swap path is 4, two intermediate tokens for the swap are determined as `fromTokenHopTokens[i]` and `toTokenHopTokens[j]`, with `i` and `j` iterating through the arrays `fromTokenHopTokens` and `toTokenHopTokens`, respectively.

```
// Construct the path through the two hop tokens
address[] memory path = new address[](4);
path[0] = _fromToken;
path[1] = fromTokenHopTokens[i];
path[2] = toTokenHopTokens[j];
path[3] = _toToken;
```

Subsequently, the function calls an internal function `calculateOutputAmount()`, which executes an external call to the function `router.getAmountsOut(_amountIn, _path)` to calculate the received amount of `_toToken`. However, an issue arises when `_fromToken = toTokenHopTokens[j]`. The problem stems from the fact that the UNI router consistently retrieves the STORAGE reserve value of the pair between two consecutive tokens `path[i]` and `path[i+1]`, without considering whether that pair has been used before. This discrepancy results in an incorrect calculation of the output amount for the swap.

For instance, if `_fromToken == toTokenHopTokens`, the path would look like this:

- `path[] = [_fromToken, fromTokenHopTokens[i], _fromToken, _toToken]`

Assuming the reserve of the UNI pair between `_fromToken` and `fromTokenHopTokens[i]` is denoted as `X` and `Y`:

- After the first swap, the reserves change to `X'` and `Y'`.
- In the second swap, when swapping from `fromTokenHopTokens[i]` to `_fromToken`, the function should use the updated reserves `X'` and `Y'`. However, due to the view nature of this function and the lack of real transaction execution, the reserves retrieved from the pair remain the old values `X` and `Y`, leading to an incorrect output.

Impact: The incorrect calculation of the received amount of `toToken`, resulting in an inaccurate determination of the best path for the swap.

BradMoonUESTC : The function `_routerSwapFromPath` in the provided code snippet is designed for executing token swap operations utilizing the `_uniSwapPath` parameter to define a swap path (`_uniSwapPath.path`). This path delineates the sequence of token conversions to be executed. A critical oversight in the current implementation is the absence of a check for duplicate addresses within the swap path. Without this validation, the presence of duplicate addresses could lead to unintended behaviors such as inaccurate amount calculations. If the swap mechanism does not inherently detect and handle such duplicates, it might inadvertently facilitate a token being swapped with itself, potentially causing financial losses due to slippage or imbalances in liquidity pool reserves. Additionally, smart contract swaps typically involve fee structures, where fees are applied at each step of the swap. Therefore, duplicates in the path could inadvertently result in the payment of unnecessary additional fees.

Recommendation

TrungOre : Consider skipping the iteration when `_fromToken == toTokenHopTokens[j]`

BradMoonUESTC : To mitigate the risks identified, it is strongly recommended to incorporate a validation mechanism

within the `_routerSwapFromPath` function to ensure the uniqueness of each token address in the swap path. This enhancement would prevent the same token from being inadvertently swapped with itself, thereby reducing the risk of financial loss and ensuring more efficient use of liquidity pools. Moreover, by eliminating unnecessary steps in the swap path, transaction fees can be minimized, leading to a more cost-effective swap process for users. Implementing this validation not only enhances the security and integrity of the token swap operation but also optimizes its financial efficiency.

Client Response

Fixed.fixed https://github.com/SoulSolidity/SoulZapV1/commit/13f3942feed78029300df41221596b6c8d921f49

APB-12:Unwithdrawable tokens permanently locked in the contract

Category	Severity	Client Response	Contributor
Logical	Low	Fixed	Yaodao

Code Reference

- code/contracts/SoulZap_UniV2.sol#L260-L276

```
260:/**
261:         * Setup swap amount0 and amount1
262:         */
263:        if (zapParams.liquidityPath.lpType == LPType.V2) {
264:            // Handle UniswapV2 Liquidity
265:            require(
266:                IUniswapV2Factory(IUniswapV2Router02(zapParams.liquidityPath.lpRouter).factory
()).getPair(
267:                    zapParams.token0,
268:                    zapParams.token1
269:                ) != address(0),
270:                "SoulZap: Pair doesn't exist"
271:            );
272:            vars.amount0In = zapParams.amountIn / 2;
273:            vars.amount1In = zapParams.amountIn / 2;
274:        } else {
275:            revert("SoulZap: LPType not supported");
276:        }
```

Description

Yaodao : The following logic will use 2 as a divisor to calculate amountIn as amount0In and amount1In. However, in solidity, division suffers from a division truncation problem.

```
        /**
         * Setup swap amount0 and amount1
         */
        if (zapParams.liquidityPath.lpType == LPType.V2) {
            // Handle UniswapV2 Liquidity
            require(
                IUniswapV2Factory(IUniswapV2Router02(zapParams.liquidityPath.lpRouter).factory()).ge
tPair(
                    zapParams.token0,
                    zapParams.token1
                ) != address(0),
                "SoulZap: Pair doesn't exist"
            );
            vars.amount0In = zapParams.amountIn / 2;
            vars.amount1In = zapParams.amountIn / 2;
        } else {
            revert("SoulZap: LPType not supported");
        }
```

As a result, when vars.mount0In is an odd number, there must be 1 token left in the contract. As the number of calls increases, more and more tokens are locked in the contract. Besides, there is no function to withdraw these tokens.

Recommendation

Yaodao : Recommend adding a function to withdraw these locked tokens.

Client Response

Fixed.Added Sweeper contract to withdraw locked tokens. Locked tokens should not happen though (except maybe the dust of 1s) https://github.com/SoulSolidity/SoulZapV1/commit/15974118a8a1857ae948e11f95df4f79031db0d4

APB-13:No Protection of Uninitialized Implementation Contracts From Attacker in SoulAccessRegistry

Category	Severity	Client Response	Contributor
Logical	Low	Fixed	parth_15

Code Reference

- code/contracts/access/SoulAccessRegistry.sol#L19-L25

```
19:contract SoulAccessRegistry is AccessControlEnumerableUpgradeable {
20:    bytes32 public constant ADMIN_ROLE = keccak256("ADMIN_ROLE");
21:
22:    // Mapping for role existence to prevent duplicates
23:    mapping(string => bool) public roleNameExists;
24:    // Array to keep track of all human-readable roles
25:    string[] private _roleNamesList;
```

Description

parth_15 : In the contracts implement Openzeppelin's UUPS model, uninitialized implementation contract can be taken over by an attacker with `initialize` function, it's recommended to invoke the `_disableInitializers` function in the constructor to prevent the implementation contract from being used by the attacker. However, the current implementation of `SoulAccessRegistry` does not call `_disableInitializers` which can lead attacker to call `initialize` function on implementation(logic) contract and destructing it. This should also be considered when upgrading the contract. Quoting from OpenZeppelin's guide of writing secure upgradeable contracts

Do not leave an implementation contract uninitialized. An uninitialized implementation contract can be taken over by an attacker, which may impact the proxy. To prevent the implementation contract from being used, you should invoke the _disableInitializers function in the constructor to automatically lock it when it is deployed:

There has been several vulnerabilities arised in past due to this.

Recommendation

parth_15 : To prevent the implementation contract from being used, you should invoke the `_disableInitializers` function in the constructor to automatically lock it when it is deployed:

```
/// @custom:oz-upgrades-unsafe-allow constructor
constructor() {
    _disableInitializers();
}
```

Client Response

Fixed.Nice one. Normally we initialize the contracts with zeros in the deployment scripts, but this is a much cleaner implementation.

Fix in: https://github.com/SoulSolidity/SoulZapV1/commit/4570fbd35fd02748491a4e73ec984403eaf28764

APB-14:lack of validation on caller provided `SwapPath.swapRouter` with fake router address

Category	Severity	Client Response	Contributor
Logical	Low	Fixed	Yaodao, parth_15, Kong7ych3, TrungOre

Code Reference

- code/contracts/ISoulZap_UniV2.sol#L34
- code/contracts/ISoulZap_UniV2.sol#L49
- code/contracts/SoulZap_UniV2.sol#L135-L186
- code/contracts/SoulZap_UniV2.sol#L180-L186
- code/contracts/SoulZap_UniV2.sol#L197
- code/contracts/SoulZap_UniV2.sol#L256-L258
- code/contracts/SoulZap_UniV2.sol#L288
- code/contracts/SoulZap_UniV2.sol#L303
- code/contracts/SoulZap_UniV2.sol#L312-L313
- code/contracts/SoulZap_UniV2.sol#L425-L454
- code/contracts/SoulZap_UniV2.sol#L425-L430
- code/contracts/SoulZap_UniV2.sol#L436-L444
- code/contracts/SoulZap_UniV2.sol#L442-L443

```
34:address swapRouter;

49:address lpRouter;

135:/// ------------------------------------------------------------------------
136:    /// Swap Functions
137:    /// ------------------------------------------------------------------------
138:
139:    /// @notice Zap single token to LP
140:    /// @param swapParams all parameters for zap
141:    /// @param feeSwapPath swap path for protocol fee
142:    function swap(
143:        SwapParams memory swapParams,
144:        SwapPath memory feeSwapPath
145:    )
146:        external
147:        payable
148:        override
149:        nonReentrant
150:        whenNotPaused
151:        verifyMsgValueAndWrap(swapParams.tokenIn, swapParams.amountIn)
152:    {
153:        if (address(swapParams.tokenIn) == address(Constants.NATIVE_ADDRESS)) {
154:            _swap(swapParams, feeSwapPath, true);
155:        } else {
156:            // No msg.value
157:            uint256 balanceBefore = _getBalance(swapParams.tokenIn);
158:            swapParams.tokenIn.safeTransferFrom(msg.sender, address(this), swapParams.amountIn);
159:            swapParams.amountIn = _getBalance(swapParams.tokenIn) - balanceBefore;
160:            _swap(swapParams, feeSwapPath, true);
161:        }
162:    }
163:
164:    /// @notice Ultimate ZAP function
165:    /// @dev Assumes tokens are already transferred to this contract.
166:    /// - whenNotPaused: Only works when not paused which also pauses all other extensions which
extend this
167:    /// @param swapParams all parameters for swap
168:    /// @param feeSwapPath swap path for protocol fee
169:    function _swap(SwapParams memory swapParams, SwapPath memory feeSwapPath, bool takeFee) inte
rnal whenNotPaused {
```

```
170:         // Verify inputs
171:         require(swapParams.amountIn > 0, "SoulZap: amountIn must be > 0");
172:         require(swapParams.to != address(0), "SoulZap: Can't swap to null address");
173:         require(swapParams.tokenOut != address(0), "SoulZap: tokenOut can't be address(0)");
174:         require(address(swapParams.tokenIn) != swapParams.tokenOut, "SoulZap: tokens can't be th
e same");
175:
176:         bool native = address(swapParams.tokenIn) == address(Constants.NATIVE_ADDRESS);
177:         if (native) swapParams.tokenIn = WNATIVE;
178:
179:         if (takeFee) {
180:             swapParams.amountIn -= _handleFee(
181:                 swapParams.tokenIn,
182:                 swapParams.amountIn,
183:                 feeSwapPath,
184:                 swapParams.deadline
185:             );
186:         }

180:swapParams.amountIn -= _handleFee(
181:                 swapParams.tokenIn,
182:                 swapParams.amountIn,
183:                 feeSwapPath,
184:                 swapParams.deadline
185:             );
186:         }

197:swapParams.tokenIn.approve(swapParams.path.swapRouter, swapParams.amountIn);

256:if (takeFee) {
257:             zapParams.amountIn -= _handleFee(zapParams.tokenIn, zapParams.amountIn, feeSwapPath,
zapParams.deadline);
258:         }

288:zapParams.tokenIn.approve(zapParams.path0.swapRouter, vars.amount0In);

303:zapParams.tokenIn.approve(zapParams.path1.swapRouter, vars.amount1In);

312:IERC20(zapParams.token0).approve(address(zapParams.liquidityPath.lpRouter), vars.amount0Out);
313:         IERC20(zapParams.token1).approve(address(zapParams.liquidityPath.lpRouter), v-
```

```
ars.amount1Out);

425:function _handleFee(
426:        IERC20 _inputToken,
427:        uint256 _inputAmount,
428:        SwapPath memory _feeSwapPath,
429:        uint256 _deadline
430:    ) private returns (uint256 inputFeeAmount) {
431:        (, uint256 feePercentage, uint256 feeDenominator, address feeCollector) = getFeeInfo();
432:        if (feePercentage == 0) {
433:            return 0;
434:        }
435:
436:        inputFeeAmount = (_inputAmount * feePercentage) / feeDenominator;
437:
438:        if (_feeSwapPath.path.length >= 2) {
439:            address outputToken = _feeSwapPath.path[_feeSwapPath.path.length - 1];
440:            require(soulFeeManager.isFeeToken(outputToken), "SoulZap: Invalid output token in fe
eSwapPath");
441:
442:            _inputToken.approve(_feeSwapPath.swapRouter, inputFeeAmount);
443:            uint256 amountOut = _routerSwapFromPath(_feeSwapPath, inputFeeAmount, feeCollector,
_deadline);
444:            _accumulateFeeVolume(amountOut);
445:        } else {
446:            /// @dev Input token is considered fee token or a token with no output route
447:            /// In order to not create a denial of service, we take any input token in this cas
e.
448:            _transferOut(_inputToken, inputFeeAmount, feeCollector, false);
449:            // Only increase fee volume if input token is a fee token
450:            if (soulFeeManager.isFeeToken(address(_inputToken))) {
451:                _accumulateFeeVolume(inputFeeAmount);
452:            }
453:        }
454:    }

425:function _handleFee(
426:        IERC20 _inputToken,
427:        uint256 _inputAmount,
428:        SwapPath memory _feeSwapPath,
429:        uint256 _deadline
```

```
430:     ) private returns (uint256 inputFeeAmount) {

436:inputFeeAmount = (_inputAmount * feePercentage) / feeDenominator;
437:
438:         if (_feeSwapPath.path.length >= 2) {
439:             address outputToken = _feeSwapPath.path[_feeSwapPath.path.length - 1];
440:             require(soulFeeManager.isFeeToken(outputToken), "SoulZap: Invalid output token in fe
eSwapPath");
441:
442:             _inputToken.approve(_feeSwapPath.swapRouter, inputFeeAmount);
443:             uint256 amountOut = _routerSwapFromPath(_feeSwapPath, inputFeeAmount, feeCollector,
_deadline);
444:             _accumulateFeeVolume(amountOut);

442:_inputToken.approve(_feeSwapPath.swapRouter, inputFeeAmount);
443:             uint256 amountOut = _routerSwapFromPath(_feeSwapPath, inputFeeAmount, feeCollector,
_deadline);
```

Description

Yaodao : In the function `swap()` , the tokens will be transferred to the contract, and then handle fees via `feeSwapPath`, and then swap the tokens to be `outputToken` via `swapParams.path.swapRouter`. Both `feeSwapPath` and `swapParams.path.swapRouter` are passed in via parameters. So the `feeSwapPath` and `swapParams.path.swapRouter` are determined by caller.

```
        if (takeFee) {
            swapParams.amountIn -= _handleFee(
                swapParams.tokenIn,
                swapParams.amountIn,
                feeSwapPath,
                swapParams.deadline
            );
        }
```

As a result, the caller can use a attack contract address for the `feeSwapPath` to non-payment of handling fees. The attack contract can pass all checks and logic easily.

```
        if (_feeSwapPath.path.length >= 2) {
            address outputToken = _feeSwapPath.path[_feeSwapPath.path.length - 1];
            require(soulFeeManager.isFeeToken(outputToken), "SoulZap: Invalid output token in feeSwa
pPath");

            _inputToken.approve(_feeSwapPath.swapRouter, inputFeeAmount);
            uint256 amountOut = _routerSwapFromPath(_feeSwapPath, inputFeeAmount, feeCollector, _dea
dline);
            _accumulateFeeVolume(amountOut);
        } else {
            /// @dev Input token is considered fee token or a token with no output route
            /// In order to not create a denial of service, we take any input token in this case.
            _transferOut(_inputToken, inputFeeAmount, feeCollector, false);
            // Only increase fee volume if input token is a fee token
            if (soulFeeManager.isFeeToken(address(_inputToken))) {
                _accumulateFeeVolume(inputFeeAmount);
            }
        }
```

parth_15 : In `SoulZap_UniV2`, `_handleFee` function is to take fees from user while doing `swap` or `zap`. This fee mechanism is for the protocol to generate revenue. Following is the function signature of `_handleFee`.

```
function _handleFee(
        IERC20 _inputToken,
        uint256 _inputAmount,
        SwapPath memory _feeSwapPath,
        uint256 _deadline
    )
```

The arguments are as follows: `_inputToken` - Token to swap or zap `_inputAmount` - Amount user wants to swap or za `_feeSwapPath` - path for converting fee to desired output token. This fee will be revenue to the protocol `_deadlin e` - time upto which the transaction will be valid

Following is function signature for `_swap` and `_zap`:

```
function _swap(SwapParams memory swapParams, SwapPath memory feeSwapPath, bool takeFee)
function _zap(ZapParams memory zapParams, SwapPath memory feeSwapPath, bool takeFee)
```

`feeSwapPath` is controlled by user. `SwapPath` struct is as follows:

```
struct SwapPath {
        address swapRouter;
        SwapType swapType;
        address[] path;
        uint256 amountOutMin;
    }
```

User can enter custom `swapRouter` which can be malicious to avoid paying fees. The following code gets executed in `_handleFee`.

```
_inputToken.approve(_feeSwapPath.swapRouter, inputFeeAmount);
 uint256 amountOut = _routerSwapFromPath(_feeSwapPath, inputFeeAmount, feeCollector, _deadline);
```

The contract gives approval to user controlled router and swap the tokens using `_feeSwapPath` to execute the swap. Now, if router is user controlled, it may not do the swap and just take fee tokens from the contract and return the transaction normally. So, user will not pay any fee and protocol's revenue is severly impaced.

Kong7ych3 : In SoulZap_UniV2 contract, users can swap tokens in Uniswap v2 via swap function, and add liquidity in Uniswap v2 via zap function. The token swap operation relies on the swapParams structure passed in by the user, and the liquidity add operation relies on the LiquidityPath structure passed in by the user. The user can customize the target Router, the amount of tokens to be swapped, and the amount of liquidity to be added in the swapParams and LiquidityPath parameters. The SoulZap_UniV2 contract then approves the tokens and amounts specified by the user for the Router contract, and swaps and adds liquidity.

Unfortunately, the protocol does not check swapRouter/lpRouter in the swapParams and LiquidityPath parameters passed in by the user. Therefore, when a user mistakenly transfers funds to the SoulZap_UniV2 contract, any user can pass in a fake swapRouter/lpRouter to make the SoulZap_UniV2 contract approve the tokens that the user mistakenly transferred into the contract to a malicious contract to steal the mistakenly transferred funds.

TrungOre : The `SoulZap_UniV2._handleFee()` function manages the calculation and transfer of protocol fees. It determines the protocol fee based on the provided input amount and the current epoch volume. Subsequently, the function authorizes the calculated fee amount to the `_feeSwapPath.swapRouter` for a swap to valid fee tokens.

```
function _handleFee(
    IERC20 _inputToken,
    uint256 _inputAmount,
    SwapPath memory _feeSwapPath,
    uint256 _deadline
) private returns (uint256 inputFeeAmount) {
    ...
    inputFeeAmount = (_inputAmount * feePercentage) / feeDenominator;

    if (_feeSwapPath.path.length >= 2) {
        address outputToken = _feeSwapPath.path[_feeSwapPath.path.length - 1];
        require(soulFeeManager.isFeeToken(outputToken), "SoulZap: Invalid output token in feeSwapPat
h");

        _inputToken.approve(_feeSwapPath.swapRouter, inputFeeAmount);
        uint256 amountOut = _routerSwapFromPath(_feeSwapPath, inputFeeAmount, feeCollector, _deadlin
e);
        _accumulateFeeVolume(amountOut);
    }

    ...
}
```

However, the `_feeSwapPath.swapRouter` parameter is user-specified in the functions `SoulZap_UniV2.swap()` and `SoulZap_UniV2.zap()`, and the internal function `_handleFee` doesn't validate whether `_feeSwapPath.swapRouter` is a legitimate UNI router. Exploiting this detail, an attacker could utilize a malicious `_feeSwapPath.swapRouter` to evade the protocol's fee. Specifically, the attacker might deploy a malevolent `UniswapV2Router02` contract featuring a malicious `swapExactTokensForTokens()` function, as illustrated below:

```
contract maliciousUniswapV2Router02 {
    address public attackerAddress;

    function swapExactTokensForTokens(
        uint amountIn,
        uint amountOutMin,
        address[] calldata path,
        address to,
        uint deadline
    ) {
        TransferHelper.safeTransferFrom(
            path[0], msg.sender, attackerAddress, amounts[0]
        );
    }
}
```

By assigning the above contract as `_feeSwapPath.swapRouter`, the `inputFeeAmount` will be transferred to the attacker's address, and the `amountOut` of the fee tokens will be `0`, resulting in a loss of fees for the protocol.

Impact:

- Protocol fund loss
- Attacker can bypass the protocol's fees

  TrungOre : The `SoulZap_UniV2._handleFee()` function oversees the calculation and transfer of protocol fees. This function invokes the internal functions `_routerSwapFromPath() -> routerSwap()`, initiating an external call to the `IUniswapV2Router02` contract for swapping to a valid fee token based on the `_feeSwapPath.path` provided by the `msg.sender`.

However, the internal function `_handleFee()` fails to verify whether the `_inputToken` is identical to `_feeSwapPath.path[0]`. This omission can result in a scenario where an attacker utilizes a different token for `_feeSwapPath.path[0]` during the swap, causing the amount of received `_feeSwapPath.path[_feeSwapPath.path.length - 1]` to be zero, resulting in a loss of fees for the protocol.

Using an alternative token for `_feeSwapPath.path[0]` implies that the attacker incurs a loss of `inputFeeAmount` in the `_feeSwapPath.path[0]` token. To mitigate this loss, the attacker can deploy arbitrary ERC20 tokens (call it as fake token), creating a UNI pair with 1,000,000 fake tokens and 1 token of `_feeSwapPath.path[1]`. By doing so, the attacker loses some worthless tokens, in turn causing a loss of fees for the protocol.

It's essential to note that the Uniswap router still pulls the `_inputToken` from `SoulZap_UniV2` to the UNI pair of `_feeSwapPath.path[0]` and `_feeSwapPath.path[1]`. This behavior is evident in the following code snippet: UniswapV2Router02.sol#L233-L235.

By setting `_feeSwapPath.path[0]` as the fake token deployed by the attacker and `_feeSwapPath.path[1] = _inputToken`, the attacker can manipulate the UNI pair's price composed of these two tokens, leading to the withdrawal of all `_inputToken` in that pool. This manipulation is possible because the attacker can mint any amount of fake tokens to execute the swap.

Impact:

- Protocol fund loss
- Users can prevent the increase of fee volume, providing a benefit to them for subsequent function calls.

Recommendation

Yaodao : Recommend recording a whitelist for the `feeSwapPath` and `swapParams.path.swapRouter` in the contract, controlled by admin.
parth_15 : This vulnerability will severy affect protocol's revenue. To mitigate this, the protocol shouldn't take user controlled `swapRouter`. Instead of doing that, make the uniswap router as state variable and use it every time to do `swap` and `zap`.
Kong7ych3 : Protocols that are not designed to save tokens from being transferred into a contract by mistake should prohibit other users from accessing funds transferred into a contract by the above means. It is recommended to check if the Routers in SwapPath and LiquidityPath are legal.
TrungOre : Consider adding a STORAGE address named `uniRouter` to store the legitimate UNISWAP router address. Validate the parameter `_feeSwapPath.swapRouter == uniRouter` when the `_handleFee()` function is invoked.
TrungOre : Consider adding a requirement to make sure `_inputToken = _feeSwapPath.path[0]` in function `_handleFee()`

Client Response

Fixed. Nice. Added whitelist validation check.
https://github.com/SoulSolidity/SoulZapV1/commit/3d62867b5ffc8ee0c801970979d3b8feedad61f3

APB-15:Uncheck return value of `token.approve()`

Category	Severity	Client Response	Contributor
Logical	Low	Acknowledged	Yaodao, ginlee

Code Reference

- code/contracts/SoulZap_UniV2.sol#L197
- code/contracts/SoulZap_UniV2.sol#L288
- code/contracts/SoulZap_UniV2.sol#L303
- code/contracts/SoulZap_UniV2.sol#L312-L313
- code/contracts/SoulZap_UniV2.sol#L442

```
197:swapParams.tokenIn.approve(swapParams.path.swapRouter, swapParams.amountIn);

288:zapParams.tokenIn.approve(zapParams.path0.swapRouter, vars.amount0In);

303:zapParams.tokenIn.approve(zapParams.path1.swapRouter, vars.amount1In);

312:IERC20(zapParams.token0).approve(address(zapParams.liquidityPath.lpRouter), vars.amount0Out);
313:        IERC20(zapParams.token1).approve(address(zapParams.liquidityPath.lpRouter), vars.amount1
Out);

442:_inputToken.approve(_feeSwapPath.swapRouter, inputFeeAmount);
```

Description

Yaodao : In the function `_swap()`, the function `swapParams.tokenIn.approve()` is called to approve the `amountIn` to the `swapRouter`.
In the function `_handleFee()`, the function `_inputToken.approve()` is called to approve the `inputFeeAmount` to the `swapRouter`.
However, the return value of the `approve()` is not checked, and the `approve()` may fail.
ginlee : Not all IERC20 implementations revert() when there's a failure in approve(). The function signature has a boolean return value and they indicate errors that way instead. By not checking the return value, operations that should have marked as failed, may potentially go through without actually approving anything

Recommendation

Yaodao : Recommend adding check logic to check the result of the `approve()`.
ginlee : It's recommend to using OpenZeppelin's SafeERC20 versions with the safeIncreaseAllowance &

safeDecreaseAllowance function that handles the return value check as well as non-standard-compliant tokens, or simply check return bool

Client Response

Acknowledged.

APB-16:Big deadline used for swap/zap functions

Category	Severity	Client Response	Contributor
Logical	Low	Fixed	parth_15, Chinmay

Code Reference

- code/contracts/SoulZap_UniV2_Lens.sol#L56

```
56:uint256 public constant DEADLINE = 20 minutes;
```

Description

parth_15 : The `DEADLINE` parameter for swapping and zapping in uniswap pool is hardcoded and set as constant with value `20 minutes`. This can cause the validators/miners of the chain to hold the transaction and execute it just before the deadline which may work against the favor of users. Especially, when the market is highly volatile and the price of the asset to be received is decreasing rapidly, miner may execute the transaction near the deadline causing the user to lose tokens. The other issue with this large hardcoded deadline is once the user sends the transaction with `deadline` equal to `20 minutes`, they will also need to wait for maximum 20 minutes to check if transaction is confirmed or not. This may cause bad trading experience to the users.
Chinmay : A deadline of 20 minutes is being used in the swap data. The problem with such a big deadline is that the best swap paths and best fee swap path that we are supplying in the swapParams is according to the market conditions of the time that we call getSwapData. But 20 minutes into the future, the conditions of the involved pools might significantly change and the user might not get the best amountOut for the time the txn is executed. Moreover, since the swap involves calculating minimum amounts from various pools and paths, this minimum amount may not be valid a few minutes into the future, leading to reverts. So it is better to use a reasonably smaller deadline and make the user execute early.

Recommendation

parth_15 : It is advisable to take this `deadline` input from users instead of hardcoding them into code. Also, favorable measures or checks can be placed to ensure that the deadline is within the permitted range to mitigate above issue.
Chinmay : May use a smaller deadline like 2 minutes or 5 minutes.

Client Response

Fixed.Thx, fixed in: https://github.com/SoulSolidity/SoulZapV1/commit/5793dfbdbacac114899982d31f2b1f90d6d9f05c

APB-17:Divide before multiply

Category	Severity	Client Response	Contributor
Logical	Low	Fixed	Yaodao

Code Reference

- code/contracts/extensions/Arrakis/lib/ArrakisMath.sol#L79-L89

```
79:vars.percentage0 =
80:             (((vars.underlying0 * 1e18) / (vars.underlying0 + vars.underlying1)) * vars.weightedP
rice0) /
81:             (vars.weightedPrice0 + vars.weightedPrice1);
82:
83:         vars.percentage1 =
84:             (((vars.underlying1 * 1e18) / (vars.underlying0 + vars.underlying1)) * vars.weightedP
rice1) /
85:             (vars.weightedPrice0 + vars.weightedPrice1);
86:
87:         amount0 =
88:             (((vars.percentage0 * 1e18) / (vars.percentage0 + vars.percentage1)) * swapRatioParam
s.inputAmount) /
89:             1e18;
```

Description

Yaodao : Performing integer division before multiplication truncates the low bits, losing the precision of the calculation.

Recommendation

Yaodao : Recommend applying multiplication before division to avoid loss of precision.

Client Response

Fixed.Updated here: https://github.com/SoulSolidity/SoulZapV1/commit/74f0ccb78296eee3fd3e2e7a22f5c97cad0509c5

APB-18:Any role can be set as Admin in SoulAccessRegistry::setRoleAdminByName()

Category	Severity	Client Response	Contributor
Logical	Low	Fixed	ravikiran_web3

Code Reference

- code/contracts/access/SoulAccessRegistry.sol#L93-L103

```
93:function setRoleAdminByName(string memory roleName, string memory adminRoleName) external onlyRol
e(ADMIN_ROLE) {
94:         _setRoleAdminFromName(roleName, adminRoleName);
95:     }
96:
97:     /// ------------------------------------------------------------------------
98:     /// Internal/Private functions
99:     /// ------------------------------------------------------------------------
100:
101:     function _setRoleAdminFromName(string memory roleName, string memory adminRoleName) internal
{
102:         _setRoleAdmin(_createRoleName(roleName), _createRoleName(adminRoleName));
103:     }
```

Description

ravikiran_web3 : In the current implementation, the setRoleAdminByName() function can set admin for any role. The reason it is low is because the role is controlled by Admin.

Example, there is no restriction to setup as below _setRoleAdminFromName("ADMIN_ROLE", "SOUL_ROLE");

In the above call, the SOUL_ROLE is the admin for "ADMIN_ROLE". This should be guarded even for human error.

Recommendation

ravikiran_web3 : Prevent administration on the Admin role. As a validation as below.

```
function setRoleAdminByName(string memory roleName, string memory adminRoleName) external onlyRole(A
DMIN_ROLE) {
         require(roleName!="ADMIN_ROLE","Not allowed");
        _setRoleAdminFromName(roleName, adminRoleName);
}
```

Client Response

Fixed.Thanks! Fixed in https://github.com/SoulSolidity/SoulZapV1/commit/20cd01b50c1e232408dafa819925e9620719f47f

APB-19:Hardcoded gas value can cause DOS

Category	Severity	Client Response	Contributor
Language Specific	Low	Acknowledged	parth_15

Code Reference

- code/contracts/utils/TransferHelper.sol#L86

```
86:(bool success, ) = to.call{value: amount, gas: 4899}("");
```

Description

parth_15 : The `_transferOut` is used to transfer ether. It uses fixed hardcoded gas of `4899`. The goal of this hardcoded gas stipend was to prevent reentrancy vulnerabilities, but this only makes sense under the assumption that gas costs are constant. EIP 1884 was included in the Istanbul hard fork. One of the changes included in EIP 1884 is an increase to the gas cost of the SLOAD operation, causing a contract's fallback function to cost more than 2300 gas.

Recommendation

parth_15 : Use `.call()` without hardcoding gas. Note that `.call()` does nothing to mitigate reentrancy attacks, so other precautions must be taken. To prevent reentrancy attacks, it is recommended that you use the checks-effects-interactions pattern.

Client Response

Acknowledged. Transferring native balances has been a constant issue in audit findings. We have evolved our method to support transfers, but prevent further code execution. If we need to update it in the future we can because this is just a periphery contract.

APB-20:Use abi.encodeCall instead of abi.encodeWithSelector in SoulZap_UniV2_Lens::getSwapData function

Category	Severity	Client Response	Contributor
Language Specific	Low	Fixed	ginlee

Code Reference

- code/contracts/SoulZap_UniV2_Lens.sol#L150
- code/contracts/SoulZap_UniV2_Lens.sol#L270

```
150:encodedTx = abi.encodeWithSelector(_SWAP_SELECTOR, swapParams, feeSwapPath);

270:encodedTx = abi.encodeWithSelector(_ZAP_SELECTOR, zapParams, feeSwapPath);
```

Description

ginlee : Since 0.8.11, abi.encodeCall provide type-safe encode utility comparing with abi.encodeWithSelector, abi.encodeCall provide type checking during compile time. For detail update please check links below: https://github.com/OpenZeppelin/openzeppelin-contracts/issues/3693 https://github.com/OpenZeppelin/openzeppelin-contracts/pull/4293

Recommendation

ginlee : it is recommended by OpenZeppelin to use abi.encodeCall instead of abi.encodeWithSelector

Client Response

Fixed.Nice, fixed here: https://github.com/SoulSolidity/SoulZapV1/commit/a00de063a1e9cf6fac639b71ef4f2c97c60775a8

APB-21:Wrong initialization of EpochVolumeTracker constructor will lead to very large epoch duration

Category	Severity	Client Response	Contributor
Language Specific	Low	Fixed	parth_15, Kong7ych3, Chinmay

Code Reference

- code/contracts/utils/EpochVolumeTracker.sol#L41
- code/contracts/utils/EpochVolumeTracker.sol#L52
- code/contracts/SoulZap_UniV2.sol#L91
- code/contracts/SoulZap_UniV2.sol#L92

```
41:constructor(uint256 __lastEpochStartTime, uint256 _epochDuration) {

52:_initialEpochStartTime = __lastEpochStartTime;

91:uint256 _epochStartTime

92:) SoulAccessManaged(_accessRegistry) EpochVolumeTracker(0, _epochStartTime) TransferHelper(_wnati
ve) {
```

Description

parth_15 : The `EpochVolumeTracker` contract's constructor is as follows:

```
constructor(uint256 __lastEpochStartTime, uint256 _epochDuration) {
        if (__lastEpochStartTime == 0) {
            /// @dev Default current epoch start time is current block timestamp
            _lastEpochStartTime = block.timestamp;
        } else {
            /// @dev Can set the current epoch start time to a past time or future for integration f
lexibility
            // If epoch start time is too far in the past past, then the epoch will start immediatel
y
            // IF epoch start time is in the future, then the epoch will not start until the epoch s
tart time
            _lastEpochStartTime = __lastEpochStartTime;
        }

        _initialEpochStartTime = __lastEpochStartTime;

        if (_epochDuration == 0) {
            /// @dev Default epoch duration is 28 days
            _EPOCH_DURATION = 28 days;
        } else {
            _EPOCH_DURATION = _epochDuration;
        }
    }
```

The first argument is `_initialEpochStartTime` and second argument is for `_EPOCH_DURATION`. If the first argument is `0`, `_initialEpochStartTime` is set to `block.timestamp`. If second argument is `0`, `_EPOCH_DURATION` is set to `28 days`. Now, initialization of constructor in `SoulZap_UniV2` is done as follows:

```
EpochVolumeTracker(0, _epochStartTime)
```

The first argument is `0` which will set `_initialEpochStartTime` to `block.timestamp` and second argument is `_epochStartTime` which will set `_EPOCH_DURATION` to `_epochStartTime`. Now, `_epochStartTime` will be close to `block.timestamp`. At the time of writing this issue, `block.timestamp` is `1701351303`. So, `_EPOCH_DURATION` will be set to `1701351303`, which is nearly `53 years`.source So, the epoch will be never ending and since the `fee` is charged based on `epoch`, `volume` will keep increasing for epoch but new epoch won't be started.

Kong7ych3 : In EpochVolumeTracker contracts, it initializes the `_lastEpochStartTime` and `_initialEpochStartTime` parameters with the `__lastEpochStartTime` parameter passed in via the constructor function. When the value of the `__lastEpochStartTime` parameter passed in by the deployer is 0, `_lastEpochStartTime` will be initialized to the current time. However, `_initialEpochStartTime` is not initialized to the current time, but uses the value of `__lastEpochStartTime`, which makes it remain 0 after initialization. This makes the `_resetEpoch` operation, as well as the `getTimeLeftInEpoch` function, calculate results that are not as expected. The `_resetEpoch` function is used to update the `_lastEpochStartTime` parameter, which theoretically differs from the previous by one `_EPOCH_DURATION` per update. However, when `_initialEpochStartTime` is 0, the updated `_lastEpochStartTime` parameter is still less than `_lastEpochStartTime + _EPOCH_DURATION`.

Chinmay : The name of this function parameter is wrong, because the EpochVolumeTracker contract's constructor takes

in the second parameter as the epoch duration whereas at the mentioned code location it is named as "epoch start time". Also, the attached comment says that it can be set to zero to start epoch tracking immediately, while that is true for the other parameter in the EpochVolumeTracker's constructor, but has nothing to do with epoch duration, the second parameter.

Recommendation

parth_15 : The fix is simple. It seems developer wants to set `_initialEpochStartTime` to `_epochStartTime` and set the `_EPOCH_DURATION` to default(i.e. 28 days). In this case, update the initialization of constructor as follows [here[((https://github.com/Secure3Audit/code_Apebond/blob/main/code/contracts/SoulZap_UniV2.sol#L92)):

```
EpochVolumeTracker(_epochStartTime, 0)
```

Kong7ych3 : It is recommended to assign the updated `_lastEpochStartTime` to the `_initialEpochStartTime` parameter instead of using `__lastEpochStartTime` in the constructor.

Please consider the fix:

```
    constructor(uint256 __lastEpochStartTime, uint256 _epochDuration) {
        if (__lastEpochStartTime == 0) {
            /// @dev Default current epoch start time is current block timestamp
            _lastEpochStartTime = block.timestamp;
        } else {
            /// @dev Can set the current epoch start time to a past time or future for integration f
lexibility
            // If epoch start time is too far in the past past, then the epoch will start immediatel
y
            // IF epoch start time is in the future, then the epoch will not start until the epoch s
tart time
            _lastEpochStartTime = __lastEpochStartTime;
        }

        _initialEpochStartTime = _lastEpochStartTime;

        if (_epochDuration == 0) {
            /// @dev Default epoch duration is 28 days
            _EPOCH_DURATION = 28 days;
        } else {
            _EPOCH_DURATION = _epochDuration;
        }
    }
```

Chinmay : Change parameter name to "_epochDuration" and also fix the comment.

Client Response

Fixed. Thx. Fixed in https://github.com/SoulSolidity/SoulZapV1/commit/30a9e0b7401ef57fdbc86055e63d58773f332a0e

APB-22:Length of addressSet should be checked SoulFeeManager::getFeeToken

Category	Severity	Client Response	Contributor
Language Specific	Low	Fixed	ravikiran_web3

Code Reference

- code/contracts/fee-manager/SoulFeeManager.sol#L185-L187
- code/contracts/SoulZap_UniV2_Lens.sol#L592-L594

```
185:function getFeeToken(uint256 index) external view override returns (address token) {
186:        return _validFeeTokens.at(index);
187:    }

592:function getHopTokenAtIndex(uint256 _index) public view returns (address) {
593:        return _hopTokens.at(_index);
594:    }
```

Description

ravikiran_web3 : EnumerableSet Library prescribes that the index should be less than length when calling at() function as below.

```
/**
     * @dev Returns the value stored at position `index` in the set. O(1).
     *
     * Note that there are no guarantees on the ordering of values inside the
     * array, and it may change when more values are added or removed.
     *
     * Requirements:
     *
     * - `index` must be strictly less than {length}.
     */
    function at(Bytes32Set storage set, uint256 index) internal view returns (bytes32) {
        return _at(set._inner, index);
    }
```

But, in the getFeeToken, the check was not implemented.

```
    function getFeeToken(uint256 index) external view override returns (address token) {
        return _validFeeTokens.at(index);
    }
```

ravikiran_web3 : Validate the index to be less than EnumerableSet.AddressSet length before accessing it

```
    function getHopTokenAtIndex(uint256 _index) public view returns (address) {
        return _hopTokens.at(_index);
    }
```

Recommendation

ravikiran_web3 : Add the check as below.

```
    function getFeeToken(uint256 index) external view override returns (address token) {
        require(index < _validFeeTokens.length(),"Index should be less than length");
        return _validFeeTokens.at(index);
    }
```

ravikiran_web3 : add a validation to check the incoming parameter to be less than length of the EnumerableSet

Client Response

FIxed.Thx! Fixed in https://github.com/SoulSolidity/SoulZapV1/commit/30a9e0b7401ef57fdbc86055e63d58773f332a0e

APB-23:Don't use block.timestamp as deadline in SoulZap_UniV2_Lens::_getSwapData function

Category	Severity	Client Response	Contributor
Logical	Low	Fixed	ginlee

Code Reference

- code/contracts/SoulZap_UniV2_Lens.sol#L232

```
232:deadline: block.timestamp + DEADLINE,
```

Description

ginlee :

```
    swapParams = ISoulZap_UniV2.SwapParams({
                // Set input token to NATIVE_ADDRESS if nativeSwap
                tokenIn: nativeSwap ? IERC20(Constants.NATIVE_ADDRESS) : IERC20(tokenIn),
                amountIn: amountIn, // Use full input amount here
                tokenOut: tokenOut,
                to: to,
                deadline: block.timestamp + DEADLINE,
                path: swapPath
            });
```

block.timestamp is being used as the deadline parameter in the _getSwapData function and DEADLINE is hardcoded, because of this, a malicious miner/sequencer can hold the transaction and execute it whenever wanted in order to acquire some profit from it

Recommendation

ginlee : Deadline should be set up by user as params instead of using hardcoded value or block.timestamp

Client Response

Fixed

APB-24:values returned by EpochVolumeTracker::getEpochVolumeInfo() are incorrect, buggy implementation

Category	Severity	Client Response	Contributor
Logical	Low	Fixed	ravikiran_web3

Code Reference

- code/contracts/utils/EpochVolumeTracker.sol#L77-L101

```
77:function getEpochVolumeInfo()
78:        public
79:        view
80:        override
81:        returns (
82:            uint256 epochVolume,
83:            uint256 lifetimeCumulativeVolume,
84:            uint256 epochStartCumulativeVolume,
85:            uint256 lastEpochStartTime,
86:            uint256 timeLeftInEpoch,
87:            uint256 epochDuration
88:        )
89:    {
90:        epochVolume = getEpochVolume();
91:        timeLeftInEpoch = getTimeLeftInEpoch();
92:        epochDuration = _EPOCH_DURATION;
93:        return (
94:            epochVolume,
95:            lifetimeCumulativeVolume,
96:            epochStartCumulativeVolume,
97:            lastEpochStartTime,
98:            timeLeftInEpoch,
99:            epochDuration
100:        );
101:    }
```

Description

ravikiran_web3 : The implementation for getEpochVolumeInfo() is buggy.

For the return value, the below the fields are not initialized at all. There are corresponding private state variables that should be returned.

Fields: lifetimeCumulativeVolume, epochStartCumulativeVolume, lastEpochStartTime,

Recommendation

ravikiran_web3 : Modify the function as below. Note that the returned values are private state level variables of the contract.

```
function getEpochVolumeInfo()
        public
        view
        override
        returns (
            uint256 epochVolume,
            uint256 lifetimeCumulativeVolume,
            uint256 epochStartCumulativeVolume,
            uint256 lastEpochStartTime,
            uint256 timeLeftInEpoch,
            uint256 epochDuration
        )
    {
        epochVolume = getEpochVolume();
        timeLeftInEpoch = getTimeLeftInEpoch();
        epochDuration = _EPOCH_DURATION;
        return (
            epochVolume, //@audit, note the below variables are state level variables.
        **    _lifetimeCumulativeVolume,
            _epochStartCumulativeVolume,
            _lastEpochStartTime,**
            timeLeftInEpoch,
            epochDuration
        );
    }
```

Client Response

Fixed,Nice find, thanks!

Revised here: https://github.com/SoulSolidity/SoulZapV1/commit/308e091a98473e21f3c9d2e3aad3b620dccbe0dc

APB-25:Project may fail to be deployed to chains not compatible with Shanghai hardfork

Category	Severity	Client Response	Contributor
Language Specific	Low	Fixed	ginlee

Code Reference

- code/contracts/SoulZap_UniV2.sol#L2
- code/contracts/SoulZap_UniV2_Lens.sol#L2
- code/contracts/extensions/Arrakis/lib/ArrakisMath.sol#L2
- code/contracts/fee-manager/SoulFeeManager.sol#L2
- code/contracts/full-versions/SoulZap_UniV2_Extended_V1.sol#L2
- code/contracts/full-versions/SoulZap_UniV2_Extended_V1_Lens.sol#L2

```
2:pragma solidity 0.8.23;

2:pragma solidity 0.8.23;

2:pragma solidity 0.8.23;

2:pragma solidity 0.8.23;

2:pragma solidity 0.8.23;

2:pragma solidity 0.8.23;
```

Description

ginlee : some of the contracts in scope have the version pragma fixed to be compiled using Solidity 0.8.23. This new version of the compiler uses the new PUSH0 opcode introduced in the Shanghai hard fork, which is now the default EVM version in the compiler and the one being currently used to compile the project.This opcode is still not supported by many chains and might be problematic for projects compiled with a version of Solidity >= 0.8.20 (when it was introduced)

Recommendation

ginlee : Switch to Solidity compiler version 0.8.19, or specify an EVM version that is compatible with all the blockchain networks intended to be supported by the protocol, alternatively, to prevent unintentionally forgetting this issue during future deployments and thus attempting to deploy on the Arbitrum chain, where this issue occurs most frequently.please

check this article for more details https://medium.com/coinmonks/push0-opcode-a-significant-update-in-the-latest-solidity-version-0-8-20-ea028668028a

Client Response

Fixed,Wow nice. That certainly seems like it should not be in a patch update to the compiler. 😅 💀 Updated https://github.com/SoulSolidity/SoulZapV1/commit/ee99d9a2453ef26d6dea91ee7f67fe1ff9fe83bc

APB-26:Absence of validation for feeSwapPath.swapRouter in the function SoulZap_UniV2._handleFee()

Category	Severity	Client Response	Contributor
Logical	Low	Fixed	Kong7ych3

Code Reference

- code/contracts/ISoulZap_UniV2.sol#L34
- code/contracts/SoulZap_UniV2.sol#L374
- code/contracts/SoulZap_UniV2.sol#L443

```
34:address swapRouter;

374:IUniswapV2Router02(router).swapExactTokensForTokens(amountIn, amountOutMin, path, _to, deadlin
e);

443:uint256 amountOut = _routerSwapFromPath(_feeSwapPath, inputFeeAmount, feeCollector, _deadline);
```

Description

Kong7ych3 : In SoulZap_UniV2 contracts, the `_handleFee` function is triggered when the user performs a swap and zap operation to pay the fee. The fee payment depends on the feeSwapPath parameter passed by the user. `_handleFee` will charge the fee based on the feeSwapPath parameter passed by the user. Therefore, a malicious user can construct the feeSwapPath parameter to bypass the fee collection operation. For example, construct `_feeSwapPath.path.length >= 2` to bring the `_handleFee` function into the if logic and spoof a user-controlled `_feeSwapPath.swapRouter` to invalidate the swapExactTokensForTokens operation. This causes the return value of `_routerSwapFromPath` to be 0, bypassing the charge.

Recommendation

Kong7ych3 : It is recommended to check that the `_feeSwapPath.swapRouter` passed by the user must be trustworthy when performing the `_handleFee` operation. This can be achieved by adding the swapRouter whitelist.

Client Response

Fixed

APB-27:Wrong comparison between amounts of tokens in _getFeeSwapPath function

Category	Severity	Client Response	Contributor
Logical	Low	Fixed	Chinmay

Code Reference

- code/contracts/SoulZap_UniV2_Lens.sol#L700

```
700:if (bestPath.amountOutMin > feeSwapPath.amountOutMin) {
```

Description

Chinmay : In the _getFeeSwapPath function, we are trying to find the best path of swapping the input token to a feeToken such that we receive maximum amountOutMin. But the mentioned line compares the amountOutMin of different fee tokens with each other, which is wrong because they may have different decimals and comparing the whole value may reset the best path to one which does not have the best value but has the highest decimals(and so the highest returned value) so that the amountOut comparison passes. This will lead to wrong calculations for the amountOutMin and fail at finding the real best swapping path.

Recommendation

Chinmay : Refactor the code to only compare same token's amounts with each other

Client Response

Fixed,Nice find! Updated in https://github.com/SoulSolidity/SoulZapV1/commit/14757c5d96d9638d05c0fc8a4fa0ffc951c947de

APB-28:Repeated calls of modifier on internal functions

Category	Severity	Client Response	Contributor
Gas Optimization	Informational	Fixed	Yaodao

Code Reference

- code/contracts/extensions/ApeBond/SoulZap_Ext_ApeBond.sol#L108
- code/contracts/SoulZap_UniV2.sol#L169

```
108:_swap(swapParams, feeSwapPath, skipFee);

169:function _swap(SwapParams memory swapParams, SwapPath memory feeSwapPath, bool takeFee) internal
whenNotPaused {
```

Description

Yaodao : The internal function `_swap()` can't be called by EOA or other contracts. And the modifier `whenNotPaused` is added to the internal function `_swap()`.

Due to the internal function `_swap()` is only called by the external functions `swap()` and `zapBond()`, and both the functions `swap()` and `zapBond()` have the modifier `whenNotPaused`, the modifier `whenNotPaused` will be called twice for the call of functions `swap()` and `zapBond()`.

Recommendation

Yaodao : Recommend removing the modifier `whenNotPaused` from the internal function `_swap()`.

Client Response

fixed.Fixed in https://github.com/SoulSolidity/SoulZapV1/commit/197043eb228e5218b46c3ead6faa0ff7676c98a5

APB-29:Use of `slot0` to get `sqrtPriceLimitX96` in ArrakisMath::pairTokensAndValue function can lead to price manipulation

Category	Severity	Client Response	Contributor
Oracle Manipulation	Informational	Acknowledged	ginlee, ravikiran_web3, BradMoonUESTC

Code Reference

- code/contracts/extensions/Arrakis/lib/ArrakisMath.sol#L134-L161
- code/contracts/extensions/Arrakis/lib/ArrakisMath.sol#L149-L160
- code/contracts/extensions/Arrakis/lib/ArrakisMath.sol#L151

```
134:function pairTokensAndValue(
135:        address token0,
136:        address token1,
137:        uint24 fee,
138:        address uniV3Factory
139:    ) internal view returns (uint256 price) {
140:        address tokenPegPair = IUniswapV3Factory(uniV3Factory).getPool(token0, token1, fee);
141:
142:        // if the address has no contract deployed, the pair doesn't exist
143:        uint256 size;
144:        assembly {
145:            size := extcodesize(tokenPegPair)
146:        }
147:        require(size != 0, "ArrakisMath: UniV3 pair not found");
148:
149:        uint256 sqrtPriceX96;
150:
151:        (sqrtPriceX96, , , , , , ) = IUniswapV3Pool(tokenPegPair).slot0();
152:
153:        uint256 token0Decimals = getTokenDecimals(token0);
154:        uint256 token1Decimals = getTokenDecimals(token1);
155:
156:        if (token1 < token0) {
157:            price = (2 ** 192) / ((sqrtPriceX96) ** 2 / uint256(10 ** (token0Decimals + 18 - tok
en1Decimals)));
158:        } else {
159:            price = ((sqrtPriceX96) ** 2) / ((2 ** 192) / uint256(10 ** (token0Decimals + 18 - t
oken1Decimals)));
160:        }
161:    }

149:uint256 sqrtPriceX96;
150:
151:        (sqrtPriceX96, , , , , , ) = IUniswapV3Pool(tokenPegPair).slot0();
152:
153:        uint256 token0Decimals = getTokenDecimals(token0);
154:        uint256 token1Decimals = getTokenDecimals(token1);
155:
156:        if (token1 < token0) {
157:            price = (2 ** 192) / ((sqrtPriceX96) ** 2 / uint256(10 ** (token0Decimals + 18 - tok
en1Decimals)));
```

```
158:         } else {
159:             price = ((sqrtPriceX96) ** 2) / ((2 ** 192) / uint256(10 ** (token0Decimals + 18 - t
oken1Decimals)));
160:         }

151:(sqrtPriceX96, , , , , , ) = IUniswapV3Pool(tokenPegPair).slot0();
```

Description

ginlee :

```
        (sqrtPriceX96, , , , , , ) = IUniswapV3Pool(tokenPegPair).slot0();
        uint256 token0Decimals = getTokenDecimals(token0);
        uint256 token1Decimals = getTokenDecimals(token1);

        if (token1 < token0) {
            price = (2 ** 192) / ((sqrtPriceX96) ** 2 / uint256(10 ** (token0Decimals + 18 - token1D
ecimals)));
        } else {
            price = ((sqrtPriceX96) ** 2) / ((2 ** 192) / uint256(10 ** (token0Decimals + 18 - token
1Decimals)));
        }
```

In ArrakisMath.sol, the functions pairTokensAndValue use UniswapV3.slot0 to get the value of sqrtPriceX96, which is used to calculate price. However, Uniswap.slot0 is the most recent data point and can be manipulated easily via MEV bots and Flashloans with sandwich attacks, manipulating sqrtPriceX96 will result in different effects on price calculations. If an attacker successfully manipulates sqrtPriceX96, it will lead to deviations from the expected price calculation. Consequently, traders will execute trades based on incorrect prices, causing transaction costs to deviate from expectations or obtaining a different quantity of tokens than anticipated.

ravikiran_web3 : using spot price from a Uniswap pool is vulnerable to price manipulation attacks, where the attacker can intentionally pump the value of his LP tokens slot0 represents the spot price which is used in pairTokensAndValue() function.

An attacker can drain the assets in the system's pools by under-collateralizing his Uniswap V3 LP tokens and manipulating the underlying Uniswap pool.

BradMoonUESTC : The smart contract code provided relies on a single external data source, Uniswap V3 pair, to determine the price of a token pair. It calculates the price using the square root of the price ratio (sqrtPriceX96) from the Uniswap V3 pool's `slot0()` function. However, this approach is vulnerable to manipulation. An attacker could distort the token pair's price within the Uniswap V3 pool via a flash loan attack or other methods that temporarily skew liquidity or trading. This singular reliance on one source for price data can result in inaccurate price reporting, create arbitrage opportunities, and potentially lead to substantial financial losses due to reliance on erroneous pricing in critical contract functions and transactions.

Recommendation

ginlee : Use a TWAP instead of slot0 to make any calculation
ravikiran_web3 : Avoid using the spot price of a Uniswap V3 pool. Instead, use TWAP from the Uniswap pool to derive the amounts of the underlying tokens of an LP token.
BradMoonUESTC : 1. **Diversify Data Sources:** Implement multiple oracles or data sources for price information to reduce reliance on a single point of failure.

2. **Integrate Sanity Checks:** Introduce sanity limits or thresholds that can trigger alerts or halt transactions if the reported price deviates significantly from historical norms or expected values.
3. **Historical Price Comparison:** Regularly compare real-time prices with historical data to detect and mitigate potential price manipulations.
4. **Enhanced Security Audits:** Conduct thorough security audits focusing on potential price manipulation vulnerabilities, especially in contracts relying on external price feeds.
5. **Community Involvement:** Engage with the broader developer and user community to stay informed about new types of attacks and emerging best practices for oracle security.

Client Response

Acknowledged. While this is a good reminder of needing to use a TWAP for on-chain calculations, this function is strictly mean't to be used for **READ ONLY**. I added an `/// @dev` comment for a reminder.
See https://github.com/SoulSolidity/SoulZapV1/commit/6014f3f0a8c626402f24d99b7cc0f35fb1149f03 The function is not used anywhere.

APB-30:modifier whenNotPaused can be omitted in _swap function

Category	Severity	Client Response	Contributor
Language Specific	Informational	Fixed	parth_15

Code Reference

- code/contracts/SoulZap_UniV2.sol#L142-L151
- code/contracts/SoulZap_UniV2.sol#L169

```
142:function swap(
143:        SwapParams memory swapParams,
144:        SwapPath memory feeSwapPath
145:    )
146:        external
147:        payable
148:        override
149:        nonReentrant
150:        whenNotPaused
151:        verifyMsgValueAndWrap(swapParams.tokenIn, swapParams.amountIn)

169:function _swap(SwapParams memory swapParams, SwapPath memory feeSwapPath, bool takeFee) internal
whenNotPaused {
```

Description

parth_15 : The internal function `_swap` uses `whenNotPaused` modifier. There is no need to use that modifier because it is only callable from `swap` function which already uses that modifier.

```
function _swap(SwapParams memory swapParams, SwapPath memory feeSwapPath, bool takeFee) internal whe
nNotPaused {
```

```
function swap(
        SwapParams memory swapParams,
        SwapPath memory feeSwapPath
    )
        external
        payable
        override
        nonReentrant
        whenNotPaused
        verifyMsgValueAndWrap(swapParams.tokenIn, swapParams.amountIn)
    {
```

Recommendation

parth_15 : Remove the `whenNotPaused` modifier from internal function `_swap`.

Client Response

Fixed

APB-31:Empty constructor not needed in abstract contract

Category	Severity	Client Response	Contributor
Language Specific	Informational	Fixed	parth_15

Code Reference

- code/contracts/extensions/ApeBond/SoulZap_Ext_ApeBond.sol#L50

```
50:constructor() {}
```

Description

parth_15 : The `SoulZap_Ext_ApeBond` is an `abstract` contract which means that it can't be initialized. It also has a constructor which is empty. It can be removed because it doesn't take any arguments and doesn't serve any purpose in the contract.

Recommendation

parth_15 : Remove the constructor since it doesn't serve any purpose.

Client Response

Fixed

APB-32:SoulFeeManager::isSoulFeeManager can be constant

Category	Severity	Client Response	Contributor
Gas Optimization	Informational	Acknowledged	ravikiran_web3

Code Reference

- code/contracts/fee-manager/SoulFeeManager.sol#L37

```
37:bool public override isSoulFeeManager = true;
```

Description

ravikiran_web3 : isSoulFeeManager is a shadow variable to the function isSoulFeeManager() of ISoulFeeManager. As the variable is public, it is rendering as implementation for function from the interface. This is a not a good practice.
As there is no function to modify the state of the above variable, declaring it as constant is more gas efficient.

Recommendation

ravikiran_web3 : bool constant IS_SOUL_FEE_MANAGER = true;
and implement the interface function as below.
function isSoulFeeManager() external view override returns (bool){ return IS_SOUL_FEE_MANAGER; }

Client Response

Acknowledged,

```
```As the variable is public, it is rendering as implementation for function from the interface. Thi
s is a not a good practice.```
```

This is not a good practice because of missing out on the gas savings, or something else?
```

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.