APEPARKSale.sol

Medium severity issues

• Owner Role / Centralized Risk

Only the owner role has authority over the functions shown below. Any compromise to the _owner account may allow the hacker to take advantage of this authority.

```
function whiteListBuyers(address[] memory _buyers) external onlyOwner() {
    for (uint i; i < _buyers.length; i++) {
        whiteListed[_buyers[i]] = true;
    }
}

    function initialize(
        address _DAOAddress,//0xCDf13985Af32f58a53804624c3DeD42EFB9b43S2
        address _alphaAPD,//0xE789CEccB5260B976EC5Bd772A1d832278415de9
        address _USDT,//0xD8691a1AB15a874E3894A8B5102C5cC9520cA8Ed
        uint _minAmount,//5000000000000000000000
        uint _maxAmount,//10000000000000000000000
        uint _toTalAmount,//500000000000000000000000
        uint _salePrice,//500000000000000000
        uint _startTimestamp//1647698400
        ) external onlyOwner() {
        DAOAddress = _DAOAddress;
        alphaAPD = _alphaAPD;
        USDT = _USDT;
        minAmount = _minAmount;
        maxAmount = _maxAmount;
        toTalAmount = _toTalAmount;
        salePrice = _salePrice;
        startTimestamp = _startTimestamp;
    }


function withdraw(address _token) external onlyOwner() {
    uint256 amount = IERC20(_token).balanceOf(address(this));
    IERC20(_token).transfer(msg.sender, amount);
}
```

✓ Recommendation

The risk describes the current project design and potentially makes iterations to improve the security operation and level of decentralization, which in most cases cannot be resolved entirely at the present stage. We advise the client to carefully manage the privileged account's private key to avoid any potential risks of being hacked. In general, we strongly recommend centralized privileges or roles in the protocol be improved via a decentralized mechanism or smart-contract-based accounts with enhanced security practices, e.g., multi-signature wallets.

Low severity issues

• Missing Emit Events

There should always be events emitted in the sensitive functions.

- ✓ whiteListBuyers
- ✓ initialize
- ✓ purchasea
- ✓ withdraw

✓ Recommendation

It is recommended emitting events for the sensitive functions .

• Missing Zero Address Validation

Addresses should be checked before assignment or external call to make sure they are not zero addresses.

- ✓ whiteListBuyers
- ✓ initialize

✓ Recommendation

We advise adding a zero-check for the passed-in address value to prevent unexpected errors.

• Unchecked Value of ERC-20 transfer()

In function withdraw,the external call to transfer of ERC20 contracts and the return value is not checked in either case.

✓ Recommendation

We advise the team to use SafeERC20 or make sure that the value returned from transfer()' is checked.

• For Loop Over Dynamic Array / optimization

When smart contracts are deployed or their associated functions are invoked, the execution of these operations always consumes a certain quantity of gas, according to the amount of computation required to accomplish them. Modifying an unknown-size array that grows in size over time can result in a Denial-of-Service attack. Simply by having an excessively huge array, users can exceed the gas limit, therefore preventing the transaction from ever succeeding.

- We can pass the array as calldata and use a local variable for the length of the array inside the function.

```
function whiteListBuyers(address[] memory _buyers) external onlyOwner() {
    for (uint i; i < _buyers.length; i++) {
        whiteListed[_buyers[i]] = true;
    }
}
```

✓ Recommendation

Avoid actions that involve looping across the entire data structure. If you really must loop over an array of unknown size, arrange for it to consume many blocks and thus multiple transactions.

• Variables That Could Be Declared as Immutable

Immutable state variables can be assigned during contract creation but will remain constant throughout the lifetime of a deployed contract. A big advantage of immutable variables is that reading them is significantly cheaper than reading from regular state variables since they will not be stored in storage.

```
address public alphaNFD;
address public DAOAddress;
address public USDT;

uint public minAmount;
uint public maxAmount;
uint public salePrice;

uint public totalAmount;
uint public saleAmount;

uint public startTimestamp;
```

✓ Recommendation

We recommend declaring these variables as immutable. Please note that the immutable keyword only works in Solidity version v0.6.5 and up.

BondDepository.sol

Medium severity issues

• Owner Role / Centralized Risk

Only the owner role has authority over the functions shown below. Any compromise to the _owner account may allow the hacker to take advantage of this authority.

- ✓ setBondTerms
- ✓ setAdjustment
- ✓ setStaking

✓ Recommendation

The risk describes the current project design and potentially makes iterations to improve the security operation and level of decentralization, which in most cases cannot be resolved entirely at the present stage. We advise the client to carefully manage the privileged account's private key to avoid any potential risks of being hacked. In general, we strongly recommend centralized privileges or roles in the protocol be improved via a decentralized mechanism or smart-contract-based accounts with enhanced security practices, e.g., multi-signature wallets.

Low severity issues

• Unchecked Value of ERC-20 transfer()

In function stakeOrSend,the external call to transfer of ERC20 contracts and the return value is not checked in either case.

✓ Recommendation

We advise the team to use SafeERC20 or make sure that the value returned from transfer()' is checked.

• Unchecked returned value

In the below external call, the returned values are not checked.

```
871         ITreasury( treasury ).deposit( _amount, principle, profit );
```

✓ Recommendation

We advise making sure that the value returned is checked.

• Missing Emit Events

There should always be events emitted in the sensitive functions.

- ✓ setInviteAddress
- ✓ setStaking
- ✓ setAdjustment
- ✓ setBondTerms
- ✓ initializeBondTerms

✓ Recommendation

It is recommended emitting events for the sensitive functions .

BondDepository.sol

Medium severity issues

• Owner Role / Centralized Risk

Only the owner role has authority over the functions shown below. Any compromise to the _owner account may allow the hacker to take advantage of this authority.

- ✓ setBondAddress
- ✓ setBounsAddress

✓ Recommendation

The risk describes the current project design and potentially makes iterations to improve the security operation and level of decentralization, which in most cases cannot be resolved entirely at the present stage. We advise the client to carefully manage the privileged account's private key to avoid any potential risks of being hacked. In general, we strongly recommend centralized privileges or roles in the protocol be improved via a decentralized mechanism or smart-contract-based accounts with enhanced security practices, e.g., multi-signature wallets.

• Potential Reentrancy Attack

A reentrancy attack can occur when the contract creates a function that makes an external call to another untrusted contract before resolving any effects. If the attacker can control the untrusted contract, they can make a recursive call back to the original function, repeating interactions that would have otherwise not run after the external call resolved the effects. Those mentioned functions are for swapping or setting liquidity, the external IERC20(token).transferFrom() functions, which cannot be fully trusted, are called before updating state variables, so there is a risk of reentrancy attack at these locations.

- ✓ depositHelper
- ✓ calAndSwap

✓ Recommendation

We recommend using the Checks-Effects-Interactions Pattern to avoid the risk of calling unknown contracts or applying OpenZeppelin ReentrancyGuard library - nonReentrant modifier for the mentioned functions to prevent reentrancy attack.

Low severity issues

• Potential Sandwich Attacks

A sandwich attack might happen when an attacker observes a transaction swapping tokens or adding liquidity without setting restrictions on slippage or minimum output amount. The attacker can manipulate the exchange rate by frontrunning (before the transaction is attacked) a transaction to purchase one of the assets and make profits by back running (after the transaction is attacked) a transaction to sell the asset after the transaction is attacked.

The following functions are called without setting restrictions on slippage or minimum output amount, so transactions triggering these functions are vulnerable to sandwich attacks, especially when the input amount is large:

- ✓ calAndSwap ➔ router.swapExactTokensForTokens(swapAmt, 0, path, address(this), block.timestamp);
- ✓ depositHelper ➔ router.addLiquidity(USDT, APD, USDT.myBalance(), APD.myBalance(), 0, 0, address(this), block.timestamp);

✓ Recommendation

We recommend setting reasonable minimum output amounts, instead of 0, based on token prices when calling the mentioned functions.

• Third Party Dependency

The contract is serving as the underlying entity to interact with one or more third party protocols. The scope of the audit treats third party entities as black boxes and assume their functional correctness. However, in the real world, third parties can be compromised and this may lead to lost or stolen assets. In addition, upgrades of third parties can possibly create severe impacts, such as increasing fees of third parties, migrating to new LP pools, etc.

- ✓ uniswapV2Router

✓ Recommendation

We understand that the business logic requires interaction with the third parties. We encourage the team to constantly monitor the statuses of third parties to mitigate the side effects when unexpected activities are observed.

• Missing Emit Events

There should always be events emitted in the sensitive functions.

- ✓ depositHelper
- ✓ calAndSwap

✓ Recommendation

It is recommended emitting events for sensitive functions.

• Missing Zero Address Validation

Addresses should be checked before assignment or external call to make sure they are not zero addresses.

```
293     constructor (address _deposit, ISwapV2Router _router, address _LPAddress,
294                     address _APD,address _USDT, ITreasury _treasury,address _bouns) {
295         deposit = _deposit;
296         APD = _APD;
297         USDT = _USDT;
298         factory = ISwapV2Factory(_router.factory());
299         router = _router;
300         LPAddress = _LPAddress;
301         treasury = _treasury;
302         bouns = _bouns;
303 
304     }
```

✓ Recommendation

We advise adding a zero-check for the passed-in address value to prevent unexpected errors.

APEPARKERC20Token.sol

Medium severity issues

• Initial Token Distribution / Centralization

In the constructor function, the deployer is minting tokens for his wallet address. All the initial supply are sent to the contract deployer when deploying the contract. This could be a centralization risk, the deployer can distribute those tokens without obtaining the consensus of the community.

```
385     constructor() ERC20("APEPARK DAO", "APD", 9) {
386         router = IDEXRouter(0x9Ac64Cc6e4415144C455BD8E4837Fea55603e5c3);
387         USDT = 0xCDf13985Af32f58a53804624c3DeD42EF89b4352;
388         pair = IDEXFactory(router.factory()).createPair(
389             USDT,
390             address(this)
391         );
392         _allowances[address(this)][address(router)] = uint256(-1);
393         _isFeeExempt[msg.sender] = true;
394         _isFeeExempt[address(this)] = true;
395 
396         DAOAddress = 0xCDf13985Af32f58a53804624c3DeD42EF89b4352;
397         bonusAddress = 0xCDf13985Af32f58a53804624c3DeD42EF89b4352;
398 
399         _mint(msg.sender,1e16);
400     }
```

✓ Recommendation

We recommend the team to be transparent regarding the initial token distribution process, and the team shall make enough efforts to restrict the access of the private key, We also advise the client to adopt Multisig, Timelock, and/or DAO in the project to manage the specific account in this case.

• Owner Role / Centralized Risk

Only the owner role has authority over the functions shown below. Any compromise to the _owner account may allow the hacker to take advantage of this authority.

- ✓ setFeeExemptListed
- ✓ setDAOAddress
- ✓ setBonusAddress
- ✓ setFeeRate

✓ Recommendation

The risk describes the current project design and potentially makes iterations to improve the security operation and level of decentralization, which in most cases cannot be resolved entirely at the present stage. We advise the client to carefully manage the privileged account's private key to avoid any potential risks of being hacked. In general, we strongly recommend centralized privileges or roles in the protocol be improved via a decentralized mechanism or smart-contract-based accounts with enhanced security practices, e.g., multi-signature wallets.

• Potential Reentrancy Attack

A reentrancy attack can occur when the contract creates a function that makes an external call to another untrusted contract before resolving any effects. If the attacker can control the untrusted contract, they can make a recursive call back to the original function, repeating interactions that would have otherwise not run after the external call resolved the effects.

- ✓ _swap

✓ Recommendation

We recommend using the Checks-Effects-Interactions Pattern to avoid the risk of calling unknown contracts or applying OpenZeppelin ReentrancyGuard library - nonReentrant modifier for the mentioned functions to prevent reentrancy attack.

Low severity issues

• Potential Sandwich Attacks

A sandwich attack might happen when an attacker observes a transaction swapping tokens or adding liquidity without setting restrictions on slippage or minimum output amount. The attacker can manipulate the exchange rate by frontrunning (before the transaction is attacked) a transaction to purchase one of the assets and make profits by back running (after the transaction is attacked) a transaction to sell the asset after the transaction is attacked.

The following functions are called without setting restrictions on slippage or minimum output amount, so transactions triggering these functions are vulnerable to sandwich attacks, especially when the input amount is large:

- ✓ _swap ➔ router.swapExactTokensForTokensSupportingFeeOnTransferTokens;

✓ Recommendation

We recommend setting reasonable minimum output amounts, instead of 0, based on token prices when calling the mentioned functions.

• Third Party Dependency

The contract is serving as the underlying entity to interact with one or more third party protocols. The scope of the audit treats third party entities as black boxes and assume their functional correctness. However, in the real world, third parties can be compromised and this may lead to lost or stolen assets. In addition, upgrades of third parties can possibly create severe impacts, such as increasing fees of third parties, migrating to new LP pools, etc.

- ✓ uniswapV2Router

✓ Recommendation

We understand that the business logic requires interaction with the third parties. We encourage the team to constantly monitor the statuses of third parties to mitigate the side effects when unexpected activities are observed.

• Missing Emit Events

There should always be events emitted in the sensitive functions.

- ✓ setFeeExemptListed
- ✓ setDAOAddress
- ✓ setBonusAddress
- ✓ setFeeRate

✓ Recommendation

It is recommended emitting events for sensitive functions.

• Missing Zero Address Validation

Addresses should be checked before assignment or external call to make sure they are not zero addresses.

- ✓ setFeeExemptListed
- ✓ setDAOAddress
- ✓ setBonusAddress

✓ Recommendation

We advise adding a zero-check for the passed-in address value to prevent unexpected errors.

RedeemHelper.sol

Medium severity issues

• Owner Role / Centralized Risk

Only the owner role has authority over the functions shown below. Any compromise to the _owner account may allow the hacker to take advantage of this authority.

- ✓ addBondContract
- ✓ removeBondContract

✓ Recommendation

The risk describes the current project design and potentially makes iterations to improve the security operation and level of decentralization, which in most cases cannot be resolved entirely at the present stage. We advise the client to carefully manage the privileged account's private key to avoid any potential risks of being hacked. In general, we strongly recommend centralized privileges or roles in the protocol be improved via a decentralized mechanism or smart-contract-based accounts with enhanced security practices, e.g., multi-signature wallets.

Low severity issues

• Missing Emit Events

There should always be events emitted in the sensitive functions.

- ✓ setFeeExemptListed
- ✓ setDAOAddress
- ✓ setBonusAddress
- ✓ setFeeRate

✓ Recommendation

It is recommended emitting events for sensitive functions.

Staking.sol

Medium severity issues

• Owner Role / Centralized Risk

Only the owner role has authority over the functions shown below. Any compromise to the _owner account may allow the hacker to take advantage of this authority.

- ✓ setLockTime
- ✓ setWarmup
- ✓ setContract
- ✓ giveLockBonus

✓ Recommendation

The risk describes the current project design and potentially makes iterations to improve the security operation and level of decentralization, which in most cases cannot be resolved entirely at the present stage. We advise the client to carefully manage the privileged account's private key to avoid any potential risks of being hacked. In general, we strongly recommend centralized privileges or roles in the protocol be improved via a decentralized mechanism or smart-contract-based accounts with enhanced security practices, e.g., multi-signature wallets.

Low severity issues

• Missing Emit Events

There should always be events emitted in the sensitive functions.

- ✓ Stake
- ✓ claim
- ✓ unLook
- ✓ unstake
- ✓ rebase
- ✓ setContract

✓ Recommendation

It is recommended emitting events for sensitive functions.

StakingDistributor.sol

Medium severity issues

• Owner Role / Centralized Risk

Only the owner role has authority over the functions shown below. Any compromise to the _owner account may allow the hacker to take advantage of this authority.

- ✓ addRecipient
- ✓ removeRecipient
- ✓ setAdjustment

✓ Recommendation

The risk describes the current project design and potentially makes iterations to improve the security operation and level of decentralization, which in most cases cannot be resolved entirely at the present stage. We advise the client to carefully manage the privileged account's private key to avoid any potential risks of being hacked. In general, we strongly recommend centralized privileges or roles in the protocol be improved via a decentralized mechanism or smart-contract-based accounts with enhanced security practices, e.g., multi-signature wallets.

Low severity issues

• Missing Emit Events

There should always be events emitted in the sensitive functions.

- ✓ Distribute
- ✓ addRecipient
- ✓ removeRecipient
- ✓ setAdjustment

✓ Recommendation

It is recommended emitting events for sensitive functions.

StakingHelper.sol

Low severity issues

• Unchecked Value of ERC-20 transferFrom()

In function stake,the external call to transferFrom of ERC20 contracts and the return value is not checked in either case.

✓ Recommendation

We advise the team to use SafeERC20 or make sure that the value returned from transferFrom()' is checked.

Treasury.sol

Low severity issues

• Unchecked Value of ERC-20 transfer()

In function incurDebt,the external call to transfer of ERC20 contracts and the return value is not checked in either case.

```
343     function incurDebt( uint _amount, address _token ) external {
344         require( isDebtor[ msg.sender ], "Not approved" );
345         require( isReserveToken[ _token ], "Not accepted" );
346 
347         uint value = valueOf( _token, _amount );
348 
349         uint maximumDebt = IERC20( sAPD ).balanceOf( msg.sender ); // Can only borrow against sAPD held
350         uint availableDebt = maximumDebt.sub( debtorBalance[ msg.sender ] );
351         require( value <= availableDebt, "Exceeds debt limit" );
352 
353         debtorBalance[ msg.sender ] = debtorBalance[ msg.sender ].add( value );
354         totalDebt = totalDebt.add( value );
355 
356         totalReserves = totalReserves.sub( value );
357         emit ReservesUpdated( totalReserves );
358 
359         IERC20( _token ).transfer( msg.sender, _amount );
360 
361         emit CreateDebt( msg.sender, _token, _amount, value );
```

✓ Recommendation

We advise the team to use SafeERC20 or make sure that the value returned from transfer()' is checked.