

B18

Security

Assessment

Aegixe Assessed on March 23, 2026





Aegix Assessed on March 23, 2026

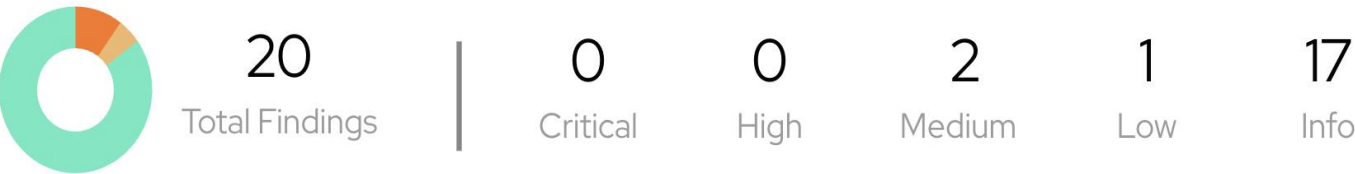
B18

The security assessment was prepared by Aegix

Executive Summary

TYPES	ECOSYSTEM	LANGUAGE	TIMELINE
A DeFi staking and settlement system deployed on Base	EVM Compatible	Solidity	inal report published on 23/03/2026

Vulnerability Summary



Critical		Directly causes loss of user or protocol funds; exploitable by any user without special privileges
High		Major fund loss or severe protocol disruption; may require specific conditions
Medium		Partial fund loss, griefing attacks, or degraded protocol functionality; may require attacker self-sacrifice
Low		Minimum-threshold bypasses, quality issues, or theoretical risks with limited direct impact
Info		The project team does not consider this a vulnerability or it aligns with the intended design

Security Audit Report (Final Version)

- B18 Protocol

Date: March 23, 2026

Auditor: Aegixe

Source code hash: B18.zip

7024140afb228a1a98803646a1eca667a967cd12c1a994172cda5d9ab2a6943c

Table of Contents

1. [Executive Summary](#)
2. [Audit Scope](#)
3. [Risk Classification](#)
4. [Detailed Findings](#)
 - [F-01 • Medium - Spot Uniswap V2 Oracle Manipulation](#)
 - [F-02 • Info - Beacon upgrade permission is incorrectly granted to `PARAMETER\_ROLE` instead of `UPGRADER\_ROLE`](#)
 - [F-03 • Info - `AccessCenter` mixes non-upgradeable `Initializable` with `AccessControlUpgradeable`](#)
 - [F-04 • Info - Creating a `0-day` vault causes permanent user fund lockup](#)
 - [F-05 • Info - `PARAMETER\_ROLE` can steal treasury funds by changing addresses through `Common`](#)
 - [F-06 • Low - Trading lock and transfer tax can be bypassed through unregistered AMM pairs](#)
 - [F-07 • Info - Vaults are permanently bound to the initial staking factory](#)
 - [F-08 • Info - Cross-contract enum drift after upgrades can misroute rewards](#)
 - [F-09 • Info - Residual router approvals can lead to token theft if the router is compromised or mismatched](#)
 - [F-10 • Info - `operationalStake` mints treasury-backed positions without USDC backing](#)
 - [F-11 • Info - USDC blacklist mechanics permanently lock funds in delayed orders](#)

- F-12 • Info - The old `feeTo` remains whitelisted after the fee recipient is changed
- F-13 • Info - Hardcoded USDC address in `PreSale` causes cross-chain payment denial of service
- F-14 • Medium - Pagination functions copy the entire storage array into memory, causing gas denial of service
- F-15 • Info - `verfiySignMessage` lacks EIP-1271 support for contract signers
- F-16 • Info - `PreSale` leaves stale approvals to multiple targets
- F-17 • Info - `PARAMETER_ROLE` can freeze arbitrary addresses through `swapPairList` before trading starts
- F-18 • Info - Hardcoded `minAmountOut = 1` in `B18SPP.swapTokens` enables sandwich attacks
- F-19 • Info - `dropVault` removes vault mappings while orders are still active
- F-20 • Info - `createOrder` allows the zero address as beneficiary

5. Additional Notes

6. Disclaimer

Executive Summary

This report is the **final version** of the security audit for the B18 protocol. It is based on the original audit report dated March 11, 2026, the project team's responses, and the post-remediation code version.

Audit Scope

Original audit files	<code>AccessCenter.sol</code> • <code>AccessControlledBase.sol</code> • <code>B18Dynamic.sol</code> • <code>B18SPP.sol</code> • <code>B18Settlement.sol</code> • <code>B18StakingFactory.sol</code> • <code>B18Token.sol</code> • <code>B18Treasury.sol</code> • <code>B18Vault.sol</code> • <code>B18VaultFactory.sol</code> • <code>Common.sol</code> • <code>ContractBase.sol</code> • <code>PreSale.sol</code> • <code>UserRegister.sol</code>
New files added in the remediated version	<code>B18Oracle.sol</code> • <code>B18MappingToken.sol</code>

Risk Classification

Severity	Definition
Critical	Directly causes loss of user or protocol funds; exploitable by any user without special privileges
High	Major fund loss or severe protocol disruption; may require specific conditions
Medium	Partial fund loss, griefing attacks, or degraded protocol functionality; may require attacker self-sacrifice
Low	Minimum-threshold bypasses, quality issues, or theoretical risks with limited direct impact
Info	The project team does not consider this a vulnerability or it aligns with the intended design

Detailed Findings

F-01 • Medium - Spot Uniswap V2 Oracle Manipulation

Location: `B18Token.convert2Token` • `B18Settlement.calcDelayedOrder` • `B18StakingFactory.lpStake` • `B18StakingFactory.treasuryStake`

Fix Status: Not fully remediated

Fix Details

The remediated version adds an independent `B18Oracle.sol` TWAP oracle contract and introduces oracle routing logic in `Common.sol`. When `oracleAddress != address(0)`, `Common.convert2USDC()` and `Common.convert2B18()` will preferentially call the TWAP oracle.

Plain Text

```
1 // Common.sol (remediated version)
2 function convert2USDC(uint256 amount) external view returns (uint256)
3 {
4     if(oracleAddress == address(0))
5     {
6         return IB18Token(tokenAddress).convert2USDC(amount); // <- stil
7     }
8     return IB18Oracle(oracleAddress).convert2USDC(amount);
9 }
```

Residual Risks

Risk 1: `B18Oracle.update()` has no minimum update interval

Plain Text

```
1 // B18Oracle.sol (remediated version)
2 function update() external onlyOracleUpdateRole
3 {
4     ...
5     uint32 timeElapsed = blockTimestamp - blockTimestampLast;
6     if(timeElapsed > 0) // <- callable every block, making the TWAP win
7     {
8         price0Average = ...
9     }
10 }
```

If an account with `ORACLE_UPDATE_ROLE` is compromised, an attacker can call `update()` at high frequency and quickly force the TWAP to converge toward the current spot price, effectively nullifying TWAP protection.

Risk 2: fallback to spot pricing when `oracleAddress == address(0)`

If `oracleAddress` is not registered through the admin function after deploying `B18Oracle`, or is reset to zero during an emergency, all critical calculations depending

on `Common.convert2USDC/B18()` will fall back to the spot oracle, fully reintroducing the original F-01 attack surface.

Risk 3: some `B18StakingFactory` paths still bypass `Common` routing

In the remediated version, `B18StakingFactory.sol` still directly calls `IB18Token(common.tokenAddress()).convert2B18/USDC()` in multiple places instead of routing through `Common.convert2B18/USDC()`, which prevents those paths from benefiting from the TWAP oracle:

Plain Text

```
1 // B18StakingFactory.sol (remediated version)
2 function singleStake(uint256 _principal, bool _isCompound, uint256 _days) external
3 {
4     uint256 usdcValue = IB18Token(common.tokenAddress()).convert2USDC(_principal); // <- still spot price
5     ...
6 }
7 function _addLiquidity(address account, uint256 usdc) private
8 {
9     uint256 b18LpAmount = IB18Token(common.tokenAddress()).convert2B18(half); // <- still spot price
10    ...
11 }
```

Risk 4: no staleness protection

`B18Oracle` does not record the last update time, and callers do not check whether the price is stale. If the `ORACLE_UPDATE_ROLE` account goes offline for an extended period, the protocol may continue using an outdated TWAP that materially deviates from market pricing.

Original Details

`convert2USDC()` and `convert2B18()` derive the B18/USDC exchange rate by reading the live reserves from the Uniswap V2 pair (`IUniswapV2Pair(mainPair).getReserves()`). There is no TWAP, no price-bound check, and no backup oracle. This spot price is directly used in all critical financial calculations throughout the protocol.

The following 8 externally callable functions inherit this risk: `withdrawalDelayedOrder`, `withdrawPrincipal`, `withdrawLevelReward`, `lpStake`, `treasuryStake`, `preSaleStake`, `singleStake`, and `singleStakeWithReward`.

Recommendation

1. Add a minimum update interval to `B18Oracle.sol` (at least 30 minutes is recommended).
2. Remove the spot-price fallback from `Common.sol` and require the oracle to be configured.
3. Standardize all price calls in `B18StakingFactory` to route through `Common.convert2B18/USDC()`.
4. Add a staleness check in `B18Oracle.consult()` (for example, revert if not updated within 2 hours).

F-02 • Info - Beacon upgrade permission is incorrectly granted to `PARAMETER_ROLE` instead of `UPGRADER_ROLE`

Location: `B18VaultFactory.upgradeVaultLogic`

Fix Status: The project team needs to retain this functionality for business reasons

Resolution Notes

In the remediated code, `B18VaultFactory.upgradeVaultLogic()` is still protected by `onlyParameterRole` and remains unchanged.

Original Details

The Beacon upgrade function in `B18VaultFactory` is protected only by `onlyParameterRole`:

Plain Text

```
1 function upgradeVaultLogic(address _newImplementation) external onlyParameterRole
2 {
3     ...
4     beacon.upgradeTo(_newImplementation);
5     ...
6 }
```

Because the Beacon owner is the factory contract, a holder of `PARAMETER_ROLE` can replace the vault logic for the entire protocol, arbitrarily modifying withdrawal paths, claim logic, accounting rules, and vault control behavior.

Impact

If `PARAMETER_ROLE` is compromised or abused, an attacker can replace the vault implementation for the entire protocol. Since this affects all active vaults, the issue is classified as informational.

Recommendation

Restrict Beacon upgrade permission to `UPGRADER_ROLE`, and add a timelock plus extra validation for approved implementation contracts.

F-03 • Info - `AccessCenter` mixes non-upgradeable `Initializable` with `AccessControlUpgradeable`

Location: `AccessCenter.initialize`

Fix Status: The project team needs to retain this functionality for business reasons

Fix Details

The remediated `AccessCenter.sol` adds `constructor() { _disableInitializers(); }` to prevent the implementation contract from being initialized directly:

Plain Text

```
1 // AccessCenter.sol (remediated version)
2 /// @custom:oz-upgrades-unsafe-allow constructor
3 constructor() {
4     _disableInitializers();
5 }
```

Because the project team has confirmed that this contract will not be upgraded, the current deployment risk is controllable.

Original Details

`AccessCenter` mixes non-upgradeable `Initializable` with `AccessControlUpgradeable`, creating an initialization context mismatch that may cause `initialize()` to revert, preventing role setup and breaking the access-control layer.

Recommendation

If upgrades are needed in the future, consistently use the OpenZeppelin upgradeable stack and inherit only upgradeable initialization base contracts compatible with `AccessControlUpgradeable`.

F-04 • Info - Creating a "0 day" vault causes permanent user fund lockup

Location: `B18VaultFactory.createVault` • `B18StakingFactory.claimInterest` • `B18StakingFactory.withdrawPrincipal`

Fix Status: The project team considers this issue ignorable

Resolution Notes

In the remediated code, `B18VaultFactory.createVault()` still does not add a `_days > 0` check and remains the same as before remediation.

Original Details

`B18VaultFactory.createVault()` does not verify that `_days` is greater than zero:

Plain Text

```
1 function createVault(uint256 _days, uint256 _simpleRate, uint256 _compoundRate) external onlyParameterRole
2 {
3     if(vaults[_days] != address(0)) revert VaultCreated(vaults[_days]);
4     // <- _days > 0 is not checked
5     bytes memory data = abi.encodeWithSelector(IB18Vault.initialize.selector, _days, ...);
6     ...
7     vaults[_days] = vaultAddr;
8     vaultDays[vaultAddr] = _days;
9 }
```

In `B18StakingFactory`, both `claimInterest()` and `withdrawPrincipal()` explicitly block `vaultDays(order.vault) == 0`:

Plain Text

```
1 if(vaultDays(order.vault) == 0) revert NotAllowed();
```

If a zero-day vault is created and later accepts deposits, all principal and interest deposited by users will become permanently unrecoverable, with no rescue path.

Impact

An operational mistake such as accidentally passing `_days = 0` can permanently lock funds.

Recommendation

Plain Text

```
1  function createVault(uint256 _days, uint256 _simpleRate, uint256 _com
   poundRate) external onlyParameterRole
2  {
3 +    if(_days == 0) revert NotAllowed();
4    if(vaults[_days] != address(0)) revert VaultCreated(vaults[_day
   s]);
5    ...
6  }
```

F-05 • Info - **PARAMETER_ROLE** can steal treasury funds by changing addresses through **Common**

Location: `Common.setStakeAddress` • `Common.setDynamicAddress` •
`B18Treasury.transferTokenToStakeFacotry` •
`B18Treasury.transferTokenToDynamicAddress`

Fix Status: The project team needs to retain this functionality for business reasons

Resolution Notes

In the remediated `Common.sol`, functions such as `setStakeAddress()` and `setDynamicAddress()` remain unchanged and are still protected by `onlyParameterRole`.

Original Details

The treasury relies on addresses stored in `Common` to determine payout destinations. These addresses are mutable by `PARAMETER_ROLE`, effectively coupling configuration control with spending authority:

Plain Text

```
1 function setDynamicAddress(address contractAddress) external onlyParameterRole
2 {
3     if(contractAddress == address(0)) revert ZeroAddress();
4     dynamicAddress = contractAddress;
5 }
```

The treasury trusts these addresses directly without additional validation:

Plain Text

```
1 function transferTokenToStakeFactory(uint256 amount) external onlyStakeFactory(msg.sender)
2 {
3     IERC20(common.tokenAddress()).safeTransfer(common.stakeAddress(), amount);
4 }
```

If `PARAMETER_ROLE` is compromised, an attacker can redirect `stakeAddress` or `dynamicAddress` to an attacker-controlled address and then invoke treasury transfer functions to drain treasury assets while still passing authorization checks.

Impact

A compromised or malicious `PARAMETER_ROLE` can drain protocol funds without requiring any additional vulnerability.

Recommendation

Separate address-configuration authority from fund-movement authority. Sensitive address changes should be timelocked and confirmed through a two-step process.

F-06 • Low - Trading lock and transfer tax can be bypassed through unregistered AMM pairs

Location: `B18Token.doTransfer`

Resolution Notes

As long as B18 trading is enabled, someone will inevitably create unofficial tax-free pools, rendering the tax mechanism ineffective. It is recommended to detect the factory origin of sender/recipient inside `doTransfer()`.

The logic of `doTransfer()` in the remediated `B18Token.sol` remains unchanged.

Original Details

Transfer logic applies launch restrictions and buy/sell tax rules only when one side of the transfer is listed in `swapPairList`:

Plain Text

```
1 function doTransfer(address sender, address recipient, uint256 amount)
  private returns (bool)
2 {
3   ...
4   if (isTaxFree == false && (swapPairList[sender] == true || swapPairList[recipient] == true))
5   {
6       if (isTraded == false) revert TradeNotStart();
7       ...
8   }
9   ...
10 }
```

On Uniswap V2-style DEXes, creating a trading pair is permissionless. An attacker can create an unlisted B18 pair, and all trading through that pair will bypass both the trading lock and taxes.

Impact

Attackers or early participants can trade before the official launch, and all trades through unofficial pairs will remain untaxed, nullifying the intended fee model.

Recommendation

Add `factory()`-based detection in `doTransfer()` to determine whether sender/recipient is a valid AMM pair from a known factory, so that all Uniswap pairs are

brought under unified control.

F-07 • Info - Vaults are permanently bound to the initial staking factory

Location: `B18Vault.initialize` • `B18VaultFactory.createVault`

Fix Status: The project team considers this issue ignorable

Resolution Notes

In the remediated `B18Vault.sol`, the `initialize()` function still permanently sets the `factory` address, with no update path.

Original Details

Each vault stores a `factory` address during initialization, and it cannot be updated later:

Plain Text

```
1 function initialize(uint256 _days, uint256 _totalSimpleRate, uint256 _totalCompoundRate, address _factory) public initializer
2 {
3     if(_factory == address(0)) revert OnlyFactory();
4     factory = _factory;
5     ...
6 }
```

`B18VaultFactory.createVault()` passes `common.stakeAddress()` into this initializer. After that, every `onlyFactory` check inside the vault depends on the original address. If governance rotates the staking contract in the future, all deployed vaults will remain bound to the old contract, causing claims and withdrawals to fail.

Impact

Replacing the staking contract may freeze all claims and withdrawals for existing vaults and orders.

Recommendation

Add a controlled factory-address update mechanism, or ensure at the design level that future staking contracts remain backward-compatible with existing vaults.

F-08 • Info - Cross-contract enum drift after upgrades can misroute rewards

Location: `B18Settlement` • `B18StakingFactory`

Fix Status: The project team considers this issue ignorable

Resolution Notes

The enum definitions in the remediated code remain unchanged.

Original Details

`B18Settlement` stores and forwards the integer value of `IB18StakingFactory.RewardType`. If enum members are reordered or new members are inserted during an upgrade, the same numeric value may be interpreted as different reward types across the two contracts, affecting cooldown periods, accounting categories, ledger debits, and token routing.

Impact

A seemingly valid upgrade can break cross-contract reward settlement logic, causing incorrect withdrawals, failed cooldown enforcement, or accounting errors.

Recommendation

Replace cross-contract enums with explicit numeric constants, or freeze the interface strictly so enum ordering remains unchanged across all upgrades.

F-09 • Info - Residual router approvals can lead to token theft if the router is compromised or mismatched

Location: `B18StakingFactory.singleStake` • `B18StakingFactory._addLiquidity`

Fix Status: The project team considers this issue ignorable

Resolution Notes

The `forceApprove` pattern in the remediated `B18StakingFactory.sol` remains unchanged.

Original Details

The staking factory uses `forceApprove` to grant token allowances to a hardcoded router, and those approvals are not reset after the operation:

Plain Text

```
1 USDC.forceApprove(address(swapRouter), usdc);
2 swapRouter.swapExactTokensForTokensSupportingFeeOnTransferTokens(...);
3 // <- approval is not reset to 0 afterward
```

Plain Text

```
1 USDC.forceApprove(address(swapRouter), half);
2 b18Token.forceApprove(address(swapRouter), b18RateLpAmount);
3 (,uint256 useUsdc,) = swapRouter.addLiquidity(...);
4 // <- residual approvals are not cleared
```

If the router is maliciously upgraded or compromised, it can use these residual approvals to transfer tokens out of the staking factory.

Impact

USDC and B18 temporarily held by the staking factory can be stolen through residual approvals, turning router trust into a direct fund-loss risk.

Recommendation

Use exact approvals per operation and immediately reset the allowance to zero afterward.

F-10 • Info - `opertionalStake` mints treasury-backed positions without USDC backing

Location: `B18StakingFactory.opertionalStake` • `B18StakingFactory._stake`

Fix Status: The project team needs to retain this functionality for business reasons

Resolution Notes

The `optionalStake()` function in the remediated `B18StakingFactory.sol` remains unchanged.

Original Details

`optionalStake` accepts any `usdc` amount, converts it into B18 principal, and then creates a treasury-backed staking position without verifying that any USDC was actually received:

Plain Text

```
1 function optionalStake(address account, uint256 usdc, bool _isCompound, uint256 _days, bool isLpStake) external onlyOperationAddress
2 {
3     if(usdc < minPrincipalValue) revert NotAllowed();
4     uint256 stakeAmount = IB18Token(common.tokenAddress()).convert2B18(usdc);
5     _stake(common.treasuryAddress(), account, stakeAmount, ...);
6 }
```

Inside `_stake`, when `from == treasury`, the contract unconditionally pulls `_principal + interest` from the treasury with no on-chain proof of funding.

Impact

A compromised or malicious `operationAddress` can fabricate treasury-funded positions without providing USDC and drain the treasury balance.

Recommendation

Require verifiable on-chain asset backing before minting treasury-backed positions, or restrict this flow to a trusted settlement pipeline with explicit inbound-funds checks.

F-11 • Info - USDC blacklist mechanics permanently lock funds in delayed orders

Location: `B18Settlement.withdrawalDelayedOrder`

Fix Status: The project team considers this issue ignorable

Resolution Notes

The push-based USDC transfer logic in the remediated `B18Settlement.sol` remains unchanged.

Original Details

`withdrawalDelayedOrder` pays users via a push transfer:

Plain Text

```
1 USDC.safeTransfer(order.account, usdcAmountAfterDeduction);
```

USDC supports address blacklisting. If `order.account` is blacklisted by Circle, every withdrawal attempt for that delayed order will revert atomically, while the corresponding B18 has already been irreversibly deducted from the staking ledger. The funds become permanently locked. There is no admin rescue path and no pull-based withdrawal alternative.

Impact

Affected users may simultaneously lose their B18 (already deducted from the ledger) and the corresponding USDC (permanently locked in Settlement).

Recommendation

Adopt a pull-payment model, or add an admin-controlled emergency rescue function with appropriate access control and timelocks, so transfer failures are decoupled from state updates.

F-12 • Info - The old `feeTo` remains whitelisted after the fee recipient is changed

Location: `B18Token._setFeeTo` • `B18Token.setFeeTo`

Fix Status: The project team considers this issue ignorable

Resolution Notes

In the remediated `B18Token.sol`, `_setFeeTo()` does not clear the whitelist status of the old address and remains the same as before remediation.

Original Details

When the fee recipient is changed, the new address is added to the whitelist, but the old address is not removed:

Plain Text

```
1 function _setFeeTo(address feeAddress) private
2 {
3     if(feeAddress == address(0))
4         revert ZeroAddress();
5
6     feeTo = feeAddress;
7     tradeWhiteList[feeTo] = true;  // <- the old feeTo is not removed f
    rom the whitelist
8 }
```

Addresses in `tradeWhiteList` are exempt from buy/sell taxes in transfers involving trading pairs. After the old fee-recipient address is replaced, it still permanently enjoys tax-free privileges unless manually revoked by an administrator.

Impact

The former fee recipient can retain privileged trading status indefinitely, weakening the intended fee model and creating unnecessary residual privilege.

Recommendation

Plain Text

```
1 function _setFeeTo(address feeAddress) private
2 {
3     if(feeAddress == address(0)) revert ZeroAddress();
4 +   if(feeTo != address(0)) tradeWhiteList[feeTo] = false;  // revok
    e whitelist for the old address
5     feeTo = feeAddress;
6     tradeWhiteList[feeTo] = true;
7 }
```

F-13 • Info - Hardcoded USDC address in `PreSale` causes cross-chain payment denial of service

Location: `PreSale.initialize`

Fix Status: The project team considers this issue ignorable

Resolution Notes

The hardcoded USDC address in the remediated `PreSale.sol` remains unchanged from the pre-remediation version.

Original Details

`PreSale` hardcodes the Base-chain USDC address during initialization:

Plain Text

```
1 function initialize(...) public initializer
2 {
3     ...
4     paymentToken = IERC20(0x833589fCD6eDb6E08f4c7C32D4f71b54bdA02913);
5     // <- hardcoded
6     ...
7 }
```

If the contract is deployed on another chain, this address may not point to the intended token contract, and the contract provides no configuration path for the payment token. The payment flow may therefore fail completely.

Impact

Incorrect deployment or future token rotation can break presale payment and settlement, causing a full denial of service for the payment path.

Recommendation

Pass the payment-token address through the initializer and validate it on the target chain.

F-14 • Info - Pagination functions copy the entire storage array into memory, causing gas denial of service

Location: `UserRegister.getUsers` • `UserRegister.getReferredUsers` • `UserRegister.pageOfAddress`

Resolution Notes

`UserRegister.sol` is still present in the remediated code and still contains the relevant pagination functions.

Original Details

The pagination helper accepts an `address[] memory` parameter, causing the full storage array to be copied before slicing:

Plain Text

```
1 function pageOfAddress(address[] memory addressList, uint256 offset, ui
   nt256 limit) internal pure returns (address[] memory)
2 {
3     uint256 addressLen = addressList.length;
4     ...
5 }
```

As the `users` array grows, gas consumption scales with the total array size rather than the requested page size, potentially causing out-of-gas failures.

Impact

Large registries may make pagination interfaces unusable for on-chain callers, resulting in denial of service for features that depend on those interfaces.

Recommendation

Operate directly on storage arrays and only copy the requested range into memory.

F-15 • Info - `verifySignMessage` lacks EIP-1271 support for contract signers

Location: `Common.verifySignMessage`

Fix Status: The project team considers this issue ignorable

Resolution Notes

In the remediated `Common.sol`, `verifySignMessage()` remains unchanged and still uses only `ECDSA.recover()`.

Original Details

Signature verification is implemented solely via `ECDSA.recover`:

Plain Text

```
1 function verifySignMessage(bytes32 dataHash, uint256 deadline, bytes ca
  lldata sign) external returns(bool)
2 {
3     ...
4     address recoverSignAddress = ECDSA.recover(dataHash, sign);
5     if(recoverSignAddress != signAddress) revert SignatureError();
6     ...
7 }
```

If governance configures `signAddress` as a multisig or another contract wallet, valid EIP-1271-style signatures cannot be verified and every flow relying on signature verification will fail.

Impact

Using a contract-based signer can unintentionally break staking or settlement paths that depend on signature verification.

Recommendation

Add EIP-1271 support for contract signers, or explicitly require `signAddress` to be an EOA and validate that in `setSignAddress()`.

F-16 • Info - `PreSale` leaves stale approvals to multiple targets

Location: `PreSale.setPresaleEnd` • `PreSale.setCompletedContract`

Fix Status: The project team considers this issue ignorable

Resolution Notes

The relevant logic in the remediated `PreSale.sol` remains unchanged.

Original Details

`setCompletedContract()` grants an allowance to the target contract but does not revoke previously granted allowances to other targets:

Plain Text

```
1 function setCompletedContract(address targetContract) external onlyParameterRole
2 {
3     if (isPresaleEnd == false) revert PresaleNotEnd();
4     if (targetContract == address(0)) revert ZeroAddress();
5     paymentToken.forceApprove(targetContract, totalCompletedPayments);
6     // <- previous approval is not revoked
7 }
```

Repeated calls can leave multiple target contracts with valid allowances to the same token balance.

Impact

Stale approvals create unnecessary risk of unauthorized token outflows, especially if an old target is later compromised.

Recommendation

Track the currently active target and revoke the previous approval before granting a new one.

F-17 • Info - `PARAMETER_ROLE` can freeze arbitrary addresses through `swapPairList` before trading starts

Location: `B18Token.setSwapPairList` • `B18Token.doTransfer`

Fix Status: The project team needs to retain this functionality for business reasons

Resolution Notes

In the remediated `B18Token.sol`, `setSwapPairList()` is now protected by `onlyOwner` instead of `onlyParameterRole`. The authority holder changed, but the function logic did not. The project team needs to retain this functionality for business reasons. `setSwapPairList` is operated by the administrator, and the related privileges will be relinquished in the future.

Original Details

`setSwapPairList()` allows any non-zero address to be marked as a swap pair:

Plain Text

```
1 function setSwapPairList(address pair, bool isPair) external onlyOwner
2 {
3     if(pair == address(0)) revert ZeroAddress();
4     if(swapPairList[pair] == isPair) revert ParameterError();
5     swapPairList[pair] = isPair;
6 }
```

Before trading starts (`isTraded == false`), `doTransfer()` reverts whenever either side of a transfer is listed in `swapPairList`. Since the function does not verify whether the address is a real AMM pair, the Owner can add any user address to the list and freeze all of that address's transfers.

Impact

This creates an admin censorship vector before launch and allows arbitrary freezing of specific accounts.

Recommendation

Verify that listed addresses are legitimate trading pairs, or separate pair tracking from transfer-blocking logic.

F-18 • Info - Hardcoded `minAmountOut = 1` in `B18SPP.swapTokens` enables sandwich attacks

Location: `B18SPP.swapTokens`

Fix Status: The project team considers this issue ignorable

Resolution Notes

The relevant logic in the remediated `B18SPP.sol` remains unchanged.

Original Details

`swapTokens` executes a Uniswap swap with `amountOutMin = 1` (effectively no slippage protection) and `deadline = block.timestamp`:

Plain Text

```
1 swapRouter.swapExactTokensForTokensSupportingFeeOnTransferTokens(  
2     amount,  
3     1,                // <- effectively no slippage protection  
4     paths,  
5     address(this),  
6     block.timestamp   // <- always passes  
7 );
```

Any observer can sandwich this transaction, and the resulting MEV loss is borne by the shared SPP fund pool.

Impact

All SPP token swaps are exposed to MEV loss. Because the function is access-controlled, severity is limited, but losses can accumulate over time.

Recommendation

Accept caller-supplied `minAmountOut` and `deadline` parameters, or compute a reasonable minimum output on-chain.

F-19 • Info - `dropVault` removes vault mappings while orders are still active

Location: `B18VaultFactory.dropVault`

Fix Status: The project team considers this issue ignorable

Resolution Notes

In the remediated `B18VaultFactory.sol`, `dropVault()` still does not check for active orders and remains unchanged from the pre-remediation version.

Original Details

`dropVault` does not verify whether the vault still contains active (unsettled) orders before deleting the mapping:

Plain Text

```
1 function dropVault(uint256 _days) external onlyParameterRole
2 {
3     address vaultAddr = vaults[_days];
4     if(vaultAddr == address(0)) revert NotAllowed();
5     vaults[_days] = address(0); // <- active orders are not checked
6     emit VaultDropCompleted(vaultAddr, _days);
7 }
```

After `dropVault` is called, `B18StakingFactory.vaultDays(order.vault)` returns `0` for all orders belonging to that vault, causing both `claimInterest()` and `withdrawPrincipal()` to revert and making the funds permanently inaccessible.

Impact

An operational error, such as accidentally calling `dropVault` on a vault with active orders, will cause all affected users to permanently lose access to their funds, with no rescue path.

Recommendation

Check the active principal balance before removing the vault:

Plain Text

```
1  function dropVault(uint256 _days) external onlyParameterRole
2  {
3      address vaultAddr = vaults[_days];
4      if(vaultAddr == address(0)) revert NotAllowed();
5 +   uint256 activePrincipal = IB18Vault(vaultAddr).totalDepositedPrin
      cipal()
6 +                                   - IB18Vault(vaultAddr).totalWithdrawnPri
      ncipal();
7 +   require(activePrincipal == 0, "Vault has active orders");
8       vaults[_days] = address(0);
9       emit VaultDropCompleted(vaultAddr, _days);
10 }
```

F-20 • Info - `createOrder` allows the zero address as beneficiary

Location: `B18Vault.createOrder`

Fix Status: The project team considers this issue ignorable

Resolution Notes

In the remediated `B18Vault.sol`, `createOrder()` still does not add a zero-address check and remains unchanged from the pre-remediation version.

Original Details

`createOrder()` does not verify that `account` is non-zero:

Plain Text

```
1 function createOrder(uint256 orderId, address account, uint256 _principal, bool isCompound) external onlyFactory returns(uint256)
2 {
3     if(_principal == 0) revert NoAmountReceived();
4     // <- account != address(0) is not checked
5     ...
6     userOrders[account].push(orderId);
7     orders[orderId] = userOrder;
8     ...
9 }
```

If the factory passes `address(0)`, the order will be stored, but it cannot later be claimed or withdrawn through normal logic, and the associated funds are permanently blackholed.

Impact

An erroneously created zero-address order will permanently lock the associated funds, with no recovery path.

Recommendation

Verify `account != address(0)` when creating the order.

Additional Notes

Permission Architecture Change: RBAC -> Ownable

The remediated version refactors the permission architecture of `B18Token` and `B18Settlement`:

Contract	Before remediation	After remediation
<code>B18Token</code>	<code>AccessControlledBase</code> (<code>onlyParameterRole</code>)	<code>Ownable</code> (<code>onlyOwner</code>)
<code>B18Settlement</code>	<code>AccessControlledBase</code>	<code>OwnableUpgradeable</code> (<code>onlyOwner</code>)

This change centralizes token-management and settlement-contract permissions into a single Owner address, simplifying the architecture but making private-key security for the Owner even more important.

Deployment Risk of the TWAP Oracle Fallback Path

`Common.convert2USDC/B18()` falls back to the spot price when `oracleAddress` is zero (Risk 2 under the residual risk section of F-01). This means that **before any staking or settlement functionality goes live, it must be confirmed that `oracleAddress` has been correctly registered through the admin function**, otherwise the protocol will depend on spot pricing and the full original F-01 attack surface will reappear.

Burn Cap Can Be Manipulated Through Direct Transfers

`B18Settlement._distributeToken` checks `b18Token.balanceOf(DEAD) > 9_100_000e18` to decide whether fees should be burned or redirected to the treasury. Because anyone can directly transfer B18 to the DEAD address, an attacker can sacrifice their own tokens to push the balance above the threshold early and permanently disable the deflationary burn mechanism. It is recommended to track protocol-initiated burns with an internal counter instead of reading `balanceOf(DEAD)`.

Disclaimer

This report reflects the auditor's (Aegixe's) review conclusions for the submitted code version, together with the auditor's overall judgment after communication with the project team. The contents of this report do not constitute a complete guarantee of the protocol's security. Additional issues not covered by this report may exist outside the audit scope, in future upgrades, or during deployment and operation. For vulnerabilities marked as "the project team needs to retain this functionality for business reasons," the residual risks are acknowledged and assumed by the project team. Final responsibility for security rests with the protocol team.

Findings Summary Table

ID	Severity	Title
F-01	Medium	Spot Uniswap V2 oracle manipulation
F-02	Info	Beacon upgrade uses <code>PARAMETER_ROLE</code> instead of <code>UPGRADER_ROLE</code>
F-03	Info	Mixed initialization base classes cause <code>AccessCenter.initialize</code> to fail
F-04	Info	Creating a zero-day vault locks user funds
F-05	Info	<code>PARAMETER_ROLE</code> changes addresses through <code>Common</code>
F-06	Low	Unregistered AMM pairs bypass trading controls
F-07	Info	Vaults are bound to the initial staking factory
F-08	Info	Cross-contract enum drift after upgrades
F-09	Info	Residual router approvals can lead to token theft
F-10	Info	<code>operationalStake</code> can mint unbacked positions
F-11	Info	USDC blacklist mechanics lock delayed orders
F-12	Info	The old <code>feeTo</code> remains whitelisted
F-13	Info	Hardcoded USDC address in <code>PreSale</code>
F-14	Medium	Pagination functions copy the entire array
F-15	Info	<code>verfiySignMessage</code> lacks EIP-1271 support
F-16	Info	<code>PreSale</code> leaves stale approvals
F-17	Info	<code>swapPairList</code> can freeze arbitrary addresses

F-18	● Info	Hardcoded <code>minAmountOut = 1</code> in SPP swaps
F-19	● Info	<code>dropVault</code> lacks an active-order check
F-20	● Info	<code>createOrder</code> allows the zero address as beneficiary

Support & Contact

For questions, assistance, or to report findings during remediation:

Email: aegixe@gmail.com

Official website: <https://aegixe.com>



<<< AEGIXE Chain Shield >>>

safeguarding web3security and protecting digital assets

Founded in 2019, Aegixe is a company focused on Web3 security. It was co-founded by cybersecurity engineers from Israel and Hong Kong with extensive experience in on-chain attack and defense, cryptographic protocols, and cross-chain security. Team members have participated in the analysis and vulnerability disclosure of numerous globally renowned security incidents and possess rich experience in auditing and conducting attack and defense operations on large-scale Web3 systems.

Aegixe currently serves hundreds of projects across various sectors, including DeFi, GameFi, RWA, L1/L2 public chains, cross-chain bridges, and payment protocols, providing the rapidly evolving Web3 industry with faster detection speeds, broader coverage, and more intelligent security capabilities.

