VELOCORE

Audit Report

Wed Aug 16 2023

contact@scalebit.xyz https://twitter.com/scalebit_

ScaleBit

VELOCORE Audit Report

1 Executive Summary

1.1 Project Information

Description	VELOCORE is a dex that incentivize veVC holders or procure veVC to strategically reallocate emissions towards their liquidity pools.
Type	Dex
Auditors	ScaleBit
Timeline	Mon Jul 17 2023 – Wed Aug 16 2023
Languages	Solidity
Platform	Linea
Methods	Architecture Review, Unit Testing, Manual Review
Source Code	https://github.com/velocore/audit
Commits	8ccd8e02714b7ace420ca6e82d84c7fc6556ae43 179d215082383454881f6475bca10ca54af77495

1.2 Files in Scope

The following are the SHA1 hashes of the original reviewed files.

ID	File	SHA-1 Hash
LEN	src/lens/Lens.sol	d94c368a954b51c13441bc5be2de4f4c562c46cd
VLE	src/lens/VelocoreLens.sol	4eeb811a43dcb11d90273260cd4b2cb3eaa32d64
NFTHF	src/NFTHolderFacet.sol	c2c52c106c3cc7c4eed471df6e003565238a9622
SAU	src/authorizer/SimpleAuthorizer.sol	8d5314930f180346754e9d4e4245f3a03baacc72
VST	src/VaultStorage.sol	6e1c1843dfddca8ae93a84498068fe9d7ef6bd14
PBL	src/lib/PoolBalanceLib.sol	32817fe2ef9e39b92faa4d9b991157ced74526ba
UME	src/lib/UncheckedMemory.sol	db9b77a013bb626de0bd4cc56877685209345fa6
RPO	src/lib/RPow.sol	184c46905583f19bbfb37fdc0e4bc7456af78d29
TOK	src/lib/Token.sol	0a3eff0bdc1e11518414b4762e7ff4d39a17615c
WETHC	src/pools/weth/WETHConverter.sol	4f111d5ac4355e5d3afe8c61decb0d65af0fd094
SAT	src/pools/Satellite.sol	dc042cdf99668a1339c5553e88d0b3e2f7816fa5
STG	src/pools/SingleTokenGauge.sol	95e6947708303ed46f2b082eee21ea3065ee1801

SUP	src/pools/SatelliteUpgradeable.sol	081dd6a3b3472ffdb32972412db485f4cb188475
POO	src/pools/Pool.sol	1efaa16a0b326f4a038b8e5b83e385ed60d80cfb
WPO	src/pools/wombat/WombatPool.sol	8f3f93a30edcab4cec0c97e456978ad5709baddc
LBF	src/pools/linear-bribe/LinearBribeFactory.sol	21ed575e32d8ce92e978a51364a2267501da0eaa
LBR	src/pools/linear-bribe/LinearBribe.sol	9021590e14f5a113b0051bd1ed27343761a64dc8
PWLPT	src/pools/PoolWithLPToken.sol	3fe6c9159520695fc8f72e232bb23710d70349ae
CPP	src/pools/constant-product/ConstantProductPool.sol	d9961af9bb2cf466b7ef8812ef0bc6d3cb67ff9f
CPPF	src/pools/constant-product/ConstantProductPoolFactory.sol	46e208f72749b746158aa53636c0d015173c1bad
CPL	src/pools/constant-product/ConstantProductLibrary.sol	39d4ccb42356072e25ecd2d64169fc9434525044
CCP	src/pools/curvecrypto/CurveCryptoPool.sol	306e6ba605a922744273a012910dd45b2cd883cb
CCPF	src/pools/curvecrypto/CurveCryptoPoolFactory.sol	d0640c7e6dd3b9e41bc2aa43b00a112c72421a6f
VC	src/pools/vc/VC.sol	11abcb9e10bf312f52ca74dba644057c9245c9f2
VVC	src/pools/vc/VeVC.sol	791b11738fa7639c2102d1071b70cfa399602c22
SFA	src/SwapFacet.sol	95b6e3ea8528c14d624b12a65f5d6711d7b8bde4

DIA	src/Diamond.yul	6b5da22bc8580c0d092df5f4e79aff2c4a826087
AFA	src/AdminFacet.sol	e4145a739a558b22fff5a56a9bed0d2174d2022c

1.3 Issue Statistic

Item	Count	Fixed	Acknowledged
Total	3	2	1
Informational	0	0	0
Minor	0	0	0
Medium	2	1	1
Major	1	1	0
Critical	0	0	0

1.4 ScaleBit Audit Breakdown

ScaleBit aims to assess repositories for security-related issues, code quality, and compliance with specifications and best practices. Possible issues our team looked for included (but are not limited to):

- Transaction-ordering dependence
- Timestamp dependence
- Integer overflow/underflow
- Number of rounding errors
- Unchecked External Call
- Unchecked CALL Return Values
- Functionality Checks
- Reentrancy
- Denial of service / logical oversights
- Access control
- Centralization of power
- Business logic issues
- Gas usage
- Fallback function usage
- tx.origin authentication
- Replay attacks
- Coding style issues

1.5 Methodology

The security team adopted the **"Testing and Automated Analysis"**, **"Code Review"** and **"Formal Verification"** strategy to perform a complete security test on the code in a way that is closest to the real attack. The main entrance and scope of security testing are stated in the conventions in the "Audit Objective", which can expand to contexts beyond the scope according to the actual testing needs. The main types of this security audit include:

(1) Testing and Automated Analysis

Items to check: state consistency / failure rollback / unit testing / value overflows / parameter verification / unhandled errors / boundary checking / coding specifications.

(2) Code Review

The code scope is illustrated in section 1.2.

(3) Audit Process

- Carry out relevant security tests on the testnet or the mainnet;
- If there are any questions during the audit process, communicate with the code owner in time. The code owners should actively cooperate (this might include providing the latest stable source code, relevant deployment scripts or methods, transaction signature scripts, exchange docking schemes, etc.);
- The necessary information during the audit process will be well documented for both the audit team and the code owner in a timely manner.

2 Summary

This report has been commissioned by VELOCORE to identify any potential issues and vulnerabilities in the source code of the VELOCORE smart contract, as well as any contract dependencies that were not part of an officially recognized library. In this audit, we have utilized various techniques, including manual code review and static analysis, to identify potential vulnerabilities and security issues.

During the audit, we have identified 3 issues of varying severity, listed below.

ID	Title	Severity	Status
PWL–1	`approve` Function Can Be Front–run Resulting in Token Theft	Medium	Fixed
AFA–1	The Proxy Contract Must Implement A Valid IDiamond Interface	Major	Fixed
SAU–1	`canPerform` Is Broken Lacking of Contract Check	Medium	Acknowledged

3 Participant Process

Here are the relevant actors with their respective abilities within the VELOCORE Smart Contract:

Owner

- Owner can upgrade the implementation address by calling the `upgrade ()` function.

Authorized user

- Authorized user can set fee amount by calling the `setFeeAmount()` function.
- Authorized user can set fee token by calling the `setFeeToken()` function.
- Authorized user can set treasury address by calling the `setTreasury()` function.
- Authorized user can add a new token to the protocol by calling the `addToken()` function.
- Authorized user can set a new fee value by calling the `setFee()` function.
- Authorized user can set a new decay rate value by calling the `setDecayRate()` function.
- Authorized user can register a new instance by calling the `register()` function.
- Authorized user can trigger the upgrade of the contract's implementation to a new version by calling the `upgradeTo()` function.
- Authorized user can update the contract's implementation and calls a function on the new implementation by calling the `upgradeToAndCall()` function.
- Authorized user can associate a given implementation address with multiple function signatures by calling the `admin_setFunctions()` function.
- Authorized user can pause or resume a specific feature or functionality within the contract by calling the `admin_pause()` function.
- Authorized user can add a new facet to the contract by calling the `admin_addFacet()` function.
- Authorized user can update the address of an authorizer contract by calling the `admin_setAuthorizer()` function.
- Authorized user can update the address of a treasury contract by calling the `admin_setTreasury()` function.

- Authorized user can execute a series of operations by calling the `execute()` function.
- Authorized user can withdraw multiple types of tokens from the contract by calling the `withdrawTokens()` function.
- Authorized user can withdraw tokens of the type ballot from the contract by calling the `withdrawTokens()` function.

User

- User can exchange tokens by calling the `execute()` function.
- User can perform operations and return their result by calling the `query()`function.
- User can perform a complex set of operations related to managing bribes and rewards by calling the `extort()` function.
- User can deploy a new constant product pool using the `deploy()` function.
- User can calculate the spot price of a pair of tokens using the `spotPrice()` function.
- User can retrieve gauge data related to users for a collection of Wombat pools using the `wombatGauges()` function.
- User can collects gauge data for a specific user from a range of canonical pool using the `canonicalPools()` function.
- User can collects gauge data for a specific user from a range of canonical pool using the `canonicalPools()` function.
- User can query gauge data for a specific user from a given pool using the `queryGauge()` function.
- User can execute various operations related to liquidity provision and token swaps within pools using the `velocore__execute()` function.

4 Findings

PWL-1 `approve` Function Can Be Front-run Resulting in Token Theft

Severity: Medium

Status: Fixed

Code Location:

src/pools/PoolWithLPToken.sol#81-87

Descriptions:

The `approve()` function has a known race condition that can lead to token theft. If a user calls the `approve()` function a second time on a spender that was already allowed, the spender can front-run the transaction and call `transferFrom()` to transfer the previous value and still receive the authorization to transfer the new value.

```
function approve(address spender, uint256 amount) public virtual returns (bool) {
    _allowance[msg.sender][spender] = amount;

    emit Approval(msg.sender, spender, amount);

    return true;
}
```

Suggestion:

Consider implementing functionality that allows a user to increase and decrease their allowance similar to Lido's implementation. This will help prevent users losing funds from front-running attacks.

```
/**
    * @notice Atomically increases the allowance granted to `_spender` by the caller by `_addedValue`.
    *
    * This is an alternative to `approve` that can be used as a mitigation for
    * problems described in:
    * https://github.com/OpenZeppelin/openzeppelin-contracts/blob/master/contracts/token/ERC20/IERC20.sol#L42
    * Emits an `Approval` event indicating the updated allowance.
    *
    * Requirements:
    *
    * - `_spender` cannot be the the zero address.
    * - the contract must not be paused.
    */
function increaseAllowance(address _spender, uint256 _addedValue) public returns (bool)
{
  _approve(msg.sender, _spender, allowances[msg.sender][_spender].add(_addedValue));
  return true;
}

/**
    * @notice Atomically decreases the allowance granted to `_spender` by the caller by `_subtractedValue`.
    *
    * This is an alternative to `approve` that can be used as a mitigation for
    * problems described in:
    * https://github.com/OpenZeppelin/openzeppelin-contracts/blob/master/contracts/token/ERC20/IERC20.sol#L42
    * Emits an `Approval` event indicating the updated allowance.
    *
    * Requirements:
    *
    * - `_spender` cannot be the zero address.
    * - `_spender` must have allowance for the caller of at least `_subtractedValue`.
    * - the contract must not be paused.
    */
function decreaseAllowance(address _spender, uint256 _subtractedValue) public returns (bool) {
  uint256 currentAllowance = allowances[msg.sender][_spender];
  require(currentAllowance >= _subtractedValue, "DECREASED_ALLOWANCE_BELOW_ZERO");
  _approve(msg.sender, _spender, currentAllowance.sub(_subtractedValue));
  return true;
}
```

Resolution:

Implement `decreaseAllowance()` and `increaseAllowance()` functions.

AFA-1 The Proxy Contract Must Implement A Valid IDiamond Interface

Severity: Major

Status: Fixed

Code Location:

src/AdminFacet.sol#81-101

Descriptions:

The provided `Diamond.yul` and `AdminFacet.sol` contracts do not fully conform to the EIP-2535 (Diamond Standard). Specifically, they lack the implementation of the `IDiamond` interface and do not emit the `DiamondCut` event when the set of functions in the diamond is modified.

The `Diamond.yul` contract is supposed to act as a general diamond proxy contract. As per EIP-2535, it should implement the IDiamond interface, which includes the diamondCut function among others.

On the other hand, the `AdminFacet.sol` contract is responsible for modifying the diamond structure. When facets are added or function selectors are assigned to implementations via the `admin_setFunctions`, `admin_addFacet` and `admin_setAuthorizer` functions, a DiamondCut event should be emitted to track these changes.

However, the `AdminFacet.sol` contract does not emit such events, which represents a departure from the specifications set out in [EIP-2535](EIP-2535).

```
    function admin_setFunctions(address implementation, bytes4[] calldata sigs) external authenticate {
        for (uint256 i = 0; i < sigs.length; i++) {
            _setFunction(sigs[i], implementation);
        }
    }

    /**
     * @dev delegatecalls the implementation's initializeFacet()
     */
    function admin_addFacet(IFacet implementation) external authenticate {
        bytes memory data = abi.encodeWithSelector(IFacet.initializeFacet.selector);
        assembly ("memory-safe") {
            let success := delegatecall(gas(), implementation, add(data, 32), mload(data), 0, 0)
            if iszero(success) { revert(0, 0) }
        }
    }
```

In EIP-2535, it states that:

All diamonds must implement the `IDiamond` interface.

During the deployment of a diamond any immutable functions and any external functions added to the diamond must be emitted in the `DiamondCut` event.

A `DiamondCut` event must be emitted any time external functions are added, replaced, or removed. This applies to all upgrades, all functions changes, at any time, whether through `diamondCut` or not.

Suggestion:

To bring the contracts into compliance with EIP-2535, you should consider the following modifications:

- Implement the `IDiamond` interface in `Diamond.yul`, including the `diamondCut` function and other required functions specified in the interface.
- Modify `AdminFacet.sol` to include the declaration and emission of the DiamondCut event. This event should be emitted whenever the `admin_setFunctions`, `admin_addFacet` and `admin_setAuthorizer` functions successfully modify the diamond's structure.

Resolution:

InspectorFacet.sol is loupe and VaultStorage.sol:_setFunction now emits the correct event.

SAU–1 `canPerform` Is Broken Lacking of Contract Check

Severity: Medium

Status: Acknowledged

Code Location:

src/authorizer/SimpleAuthorizer.sol#12–14

Descriptions:

As comments suggested, canPerformshould check certains roles can perform actions in specific contract. However, the code implementations lack of the check for that contract. Making the roles granted for certain actions have ability to execute those actions across the contracts.

```
interface IAuthorizer {
    /**
     * @dev Returns true if `account` can perform the action described by `actionId` in the
contract `where`.
     */
    function canPerform(bytes32 actionId, address account, address where) external view
returns (bool);
}
```

```
function canPerform(bytes32 actionId, address account, address where) external view
override returns (bool) {
    return hasRole(DEFAULT_ADMIN_ROLE, account) || hasRole(actionId, account);
}
```

Thus, a malicious role would be able to execute some actions out of expectations and bypass the checks.

```
    modifier authenticate() {
        require(

IAuthorizer(address(uint160(uint256(_readVaultStorage(SSLOT_HYPERCORE_AUTHORIZER)
                keccak256(abi.encodePacked(bytes32(uint256(uint160(factory))), msg.sig)),
msg.sender, address(this)
            ),
            "unauthorized"
        );
        _;
    }
```

```
function authenticateCaller() internal {
    bytes32 actionId =
keccak256(abi.encodePacked(bytes32(uint256(uint160(address(this)))), msg.sig));
    require(

IAuthorizer(StorageSlot.getAddressSlot(SSLOT_HYPERCORE_AUTHORIZER).value).canPerfo
            actionId, msg.sender, address(this)
        ),
        "unauthorized"
    );
}
```

Suggestion:

Either change the `canPerformcode` to check the `where` address or make it explicit that roles can perform actions across the contracts.

Resolution:

Acknowledged by Velocore dev, that `where` is indented not to check, but left to be used in the future.

Appendix 1

Issue Level

- **Informational** issues are often recommendations to improve the style of the code or to optimize code that does not affect the overall functionality.

- **Minor** issues are general suggestions relevant to best practices and readability. They don't post any direct risk. Developers are encouraged to fix them.

- **Medium** issues are non-exploitable problems and not security vulnerabilities. They should be fixed unless there is a specific reason not to.

- **Major** issues are security vulnerabilities. They put a portion of users' sensitive information at risk, and often are not directly exploitable. All major issues should be fixed.

- **Critical** issues are directly exploitable security vulnerabilities. They put users' sensitive information at risk. All critical issues should be fixed.

Issue Status

- **Fixed:** The issue has been resolved.

- **Partially Fixed:** The issue has been partially resolved.

- **Acknowledged:** The issue has been acknowledged by the code owner, and the code owner confirms it's as designed, and decides to keep it.

Appendix 2

Disclaimer

This report is based on the scope of materials and documents provided, with a limited review at the time provided. Results may not be complete and do not include all vulnerabilities. The review and this report are provided on an as–is, where–is, and as–available basis. You agree that your access and/or use, including but not limited to any associated services, products, protocols, platforms, content, and materials, will be at your own risk. A report does not imply an endorsement of any particular project or team, nor does it guarantee its security. These reports should not be relied upon in any way by any third party, including for the purpose of making any decision to buy or sell products, services, or any other assets. TO THE FULLEST EXTENT PERMITTED BY LAW, WE DISCLAIM ALL WARRANTIES, EXPRESS OR IMPLIED, IN CONNECTION WITH THIS REPORT, ITS CONTENT, RELATED SERVICES AND PRODUCTS, AND YOUR USE, INCLUDING BUT NOT LIMITED TO THE IMPLIED WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE, NOT INFRINGEMENT.

SCALEBIT VERIFIED

SCALEBIT VERIFIED