Everclear Tokenomics

November 2024

Made with ❤ by the following Creed authors: Valentin Quelquejay and Dominik Muhs

Table of Contents

Executive Summary

This report presents the results of our engagement with Everclear to review the proposed Tokenomics smart contract protocol changes.

Valentin Quelquejay and Dominik Muhs conducted the review over three weeks, from October 21, 2024, to November 29, 2024. A preliminary report was delivered on October 25. Furthermore, the assessment team conducted a follow-up review from November 30 to December 2, 2024. Overall, 34 person-days were spent.

The system consists of a set of Spokes deployed on the different chains Everclear operates on and a hub deployed on the clearing chain. Users interact with the system via the different Spoke contracts and can lock NEXT/CLEAR tokens to receive `vbCLEAR` in return, which provides them with governance rights and a share of the protocol fees.

Users can vote on which domains to allocate NEXT/CLEAR rewards. All the state related to `vbNEXT` and voting is stored on the clearing chain. Rewards are claimed from the clearing chain and bridged back to a Spoke domain to be transferred to the user. The cross-chain messaging between the spoke domains and the clearing chain is secured and handled via Hyperlane.

Two medium-severity findings regarding the base `Bridge` smart contract and potential address type incompatibilities on the clearing chain have been identified. Furthermore, the assessment team identified seven minor severity issues to improve the code's maintainability and fix potential attack vectors.

Scope and Objectives

Our initial review focused on the commit hash `3958879c40f0c35c6741e9449173c159dca20ac1`.

During the follow-up review, the scope was extended by the following items:

- `audit/creed-vbCLEAR-fixes` at `a0b06a22788c9bb1d87d65d88c4169f326f37c7c`
- `audit/creed-reward-array-check` at `788cd2cda6837d34429d4bfb45a3da4d719ba0eb`
- `audit/creed-redundant-logic` at `a1c406be2a45df41c18774d531d6b5267028d579`
- `audit/creed-lax-address-validation` at `495bab94ed0ffa941fe37bc0c142151723ef8939`
- `audit/creed-inconsistent-naming` at `2135cb9be820d69617faf5aa00bd161a1bcc100b`
- `audit/creed-eth-transfer` at `c80454c33e0984199e866f25a3cf5ab0fd76a50a`
- `audit/creed-addres-compatability` at `ce5037876709f0d77b124046c4969929b17098cd`

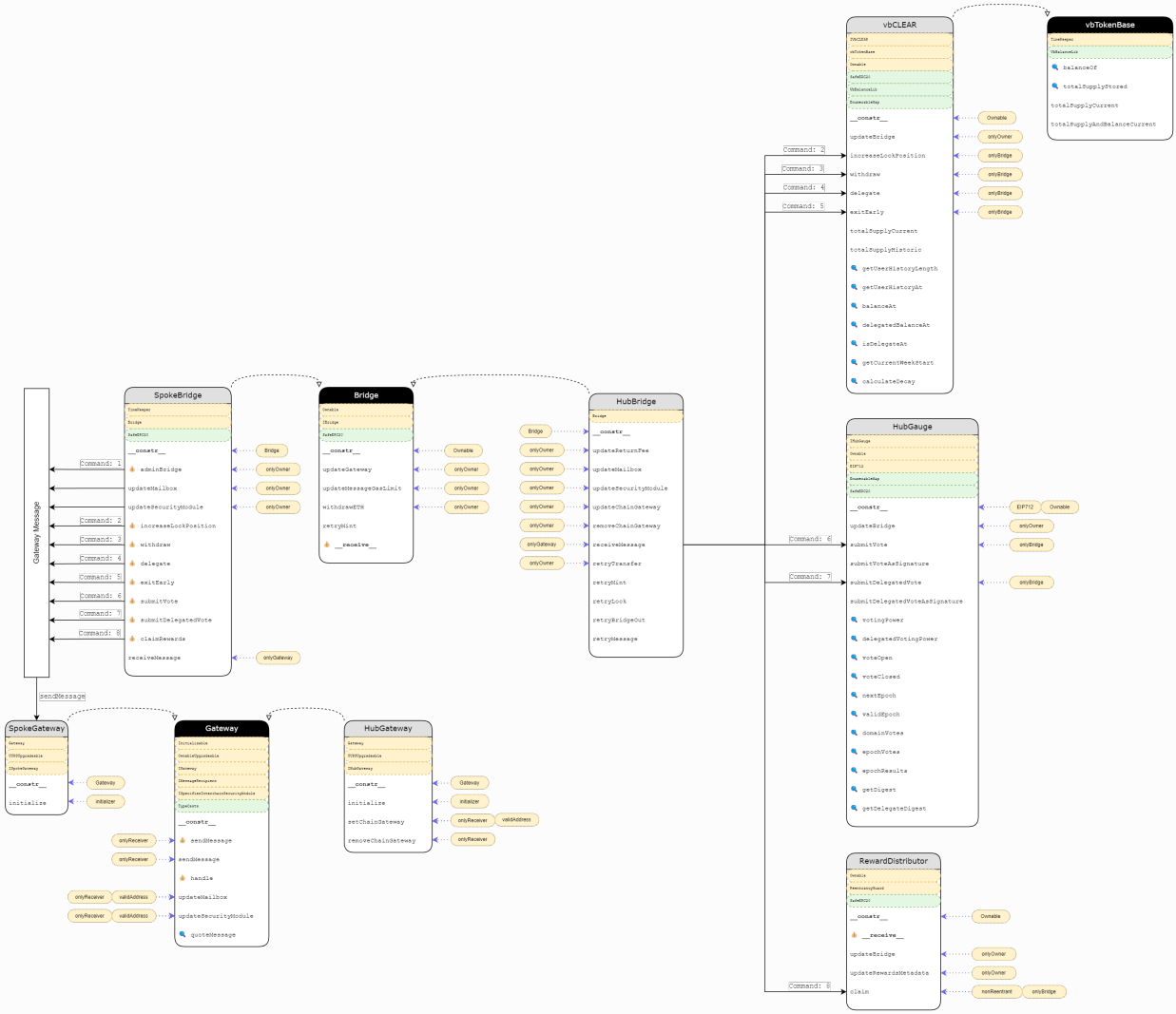
Furthermore, we reviewed minor changes to the Everclear xERC20 token contract, located under the `clear-upgrade` branch at commit `5afae395fa7ba2f9428aa92d9db6b98b083f3e00`. On the main Everclear Chimera code base, a small security-relevant finding has also been mitigated and reviewed, located on the `upgrade-destination-array` branch at commit `a867c9378451871b32af5fb1565879868a70fa03`.

Together with the Everclear team, we identified the following priorities for our review:

- Consider the system's cross-chain compatibility and flag potential integration issues.
- Review risk related to smart contract upgrades and breaking changes.
- Ensure the system's voting mechanism and delegation are correct and cannot be manipulated.
- Ensure the system is implemented consistently with the intended functionality and without unintended edge cases.
- Identify known vulnerabilities particular to smart contract systems, as outlined in our Smart Contract Security Field Guide and the ones outlined in the EEA EthTrust Security Levels Specification.

Audit Artifacts

System Architecture Diagram



Fuzzing Artifacts

As part of the audit, Creed developed several Foundry fuzz tests targeting libraries to validate key invariants of the codebase. Minor issues were identified. Additionally, a Diligence Fuzzing campaign was run for 13 hours as part of the engagement. The fuzzing harness code has been attached in Appendix A. The findings have been incorporated into the report.

Findings

Medium `transfer` May Fail For Smart Contract Wallets

Fixed

> Fixed in commit 4f705af9097c5243c7284f3019c870994d54bfc5.

The `withdrawETH` function in `Bridge` uses `transfer` to send ETH, which is limited by a 2300 gas stipend. This can cause failures when interacting with smart contract wallets that require more gas to execute their fallback or receive functions.

src/common/Bridge.sol

```
65 function withdrawETH(address _receiver) external onlyOwner {
66     payable(_receiver).transfer(address(this).balance);
67     emit WithdrawETH(_receiver, address(this).balance);
68 }
```

Recommendation

Use `call` instead of `transfer` to ensure compatibility with smart contract wallets.

Medium Potential Address Type Incompatibilities

Fixed

> The client addressed the issue via PR #1 at the following commit hash: `56c0d50fcd7c6dfc83100159a7a5156d30508074`.

In its first release, Everclear's Chimera introduced smart contracts on the Hub domain. For cross-chain-related data structures, instead of an `address` type, `bytes32` has been used to account for future integrations with spoke domains that use addresses longer than 160 bits.

The tokenomics upgrade introduces an additional set of smart contracts on the Hub domain, which may break compatibility by directly using the `address` type.

Examples

src/hub/vbCLEAR.sol

```
40 mapping(address _user => address) public voteDelegate;
41
42 /// @notice Mapping to store the relayers
43 mapping(address => bool) public isRelayer;
```

src/hub/HubGauge.sol

```
47 mapping(address _voter => mapping(uint256 _epoch => bool)) public voteReceived;
```

src/hub/HubGauge.sol

```
93 function submitVote(address _sender, uint256 _domain) external onlyBridge {
94     uint256 _epoch = _checkVotingOpen();
```

src/hub/HubGauge.sol

```
125 function submitDelegatedVote(address _sender, address[] calldata _grantors, uint…
126     uint256 _epoch = _checkVotingOpen();
```

src/hub/HubGauge.sol

```
144 function submitDelegatedVoteAsSignature(address[] calldata _grantors, uint256 _d…
```

Recommendation

We recommend reviewing the code base for user-related addresses passed as parameters and stored using the `address` type and refactoring these occurrences to `bytes32`. As a rule of thumb, all spoke-related data regarding user (and thus voter) and asset addresses should be handled using

`bytes32`, as is already done in, e.g., the `EverclearHub` contract. This refactoring will ensure maximum compatibility with the already existing smart contracts that Chimera introduced.

Minor Potential Overflows And Rounding Errors In `VbBalanceLib`

Partially Fixed

> The client partially fixed the issue in commit a0b06a22788c9bb1d87d65d88c4169f326f37c7c. Specifically, overflows in the `getValueAt` and `isExpired` functions were addressed.

The functions `getValueAt` and `isExpired` in `VbBalanceLib` do not handle potential overflows in their arithmetic operations. Specifically, if `slope * timestamp > type(uint128).max`, the computation `if (a.slope * t > a.bias)` can overflow as it is performed in `uint128` rather than `uint256`. This causes the functions to revert instead of returning the expected value (e.g., `0` for `getValueAt`). This behavior may not be critical in practice but could lead to unexpected reverts under specific conditions.

src/libs/vbBalanceLib.sol

```
61 if (a.slope * t > a.bias) {
62     return 0;
63 }
```

src/libs/vbBalanceLib.sol

```
51 function isExpired(VbBalance memory a) internal view returns (bool) {
52     return a.slope * uint128(block.timestamp) >= a.bias;
```

This might also be an issue in the `add` function:

src/libs/vbBalanceLib.sol

```
36 function add(VbBalance memory a, VbBalance memory b) internal pure returns (VbBa…
37     res.bias = a.bias + b.bias;
38     res.slope = a.slope + b.slope;
39 }
```

Additionally, `convertToVbBalance` will return a position with value `zero` for any locked position where `position.amount < MAX_LOCK_TIME`. As long as the `vbCLEAR` token has enough decimals, it should be fine given 730 days ~= 10e7, so the amount should be negligible.

src/libs/vbBalanceLib.sol

```
72 function convertToVbBalance(LockedPosition memory position) internal pure return…
73     res.slope = position.amount / MAX_LOCK_TIME;
74     res.bias = res.slope * position.expiry;
```

Recommendation

Consider using `uint256` for intermediate calculations or add a notice in the NatSpec documentation.

Minor Inconsistent Naming Of vbCLEAR And vbNEXT

Fixed

> This finding has been fixed by changing the code base's token references to CLEAR-based names in the following commit hash `2135cb9be820d69617faf5aa00bd161a1bcc100b` on the `audit/creed-inconsistent-naming` branch.

Actions are performed using the `vbCLEAR` token across the code base. However, the token contract's naming is inconsistently split between `vbCLEAR` and `vbNEXT`, decreasing the readability of the code base and its documentation and increasing the risk of human error.

Examples

src/hub/HubBridge.sol

```
150 /**
151  * @notice Retry increaseLockPosition action burning xCLEAR and locking for vbNE…
152  * @param _errorId id related to the stored error
153  */
```

src/hub/HubBridge.sol

```
165 /**
166  * @notice Retry locking for vbNEXT
167  * @param _errorId id related to the stored error
168  * @param _expiry new expiry for the lock
169  */
```

src/hub/HubGauge.sol

```
37 /// @notice The vbNEXT contract (vote locked contract)
38 IVbCLEAR public immutable vbNEXT;
```

src/hub/HubGauge.sol

```
64 constructor(string memory _name, string memory _version, address _vbNEXT, uint25…
65     Ownable(msg.sender)
66     EIP712(_name, _version)
67 {
68     if (_vbNEXT == address(0)) revert ZeroAddress();
69     if (_genesisEpoch == 0) revert ZeroValue();
70
71     vbNEXT = IVbCLEAR(_vbNEXT);
72     genesisEpoch = _genesisEpoch;
73     bridge = _bridge;
74 }
```

src/hub/HubGauge.sol

```
293 /**
294  * @notice Queries vbNEXT for the voting power of a user at a specific time
```

```
295  * @dev snapshotTime will be nextEpoch - SNAPSHOT_DEDUCTION i.e. previous timest…
296  * @param _user address voting power desired for
297  * @param _snapshotTime time of vote power
298  * @return voting power for user
299  */
300 function _votingPower(address _user, uint256 _snapshotTime) internal view return…
301     return vbNEXT.balanceAt(_user, _snapshotTime);
302 }
```

src/hub/HubGauge.sol

```
304 /**
305  * @notice Queries vbNEXT for the voting power of a delegate at a specific time …
306  * @dev snapshotTime will be nextEpoch - SNAPSHOT_DEDUCTION i.e. previous timest…
307  * @param _delegate delegate voting power is being queried for
308  * @param _grantors addresses provided that have delegated to caller
309  * @param _snapshotTime time of vote power
310  * @return delegated voting power for grantor
311  */
312 function _delegatedPower(address _delegate, address[] calldata _grantors, uint25…
313     internal
314     view
315     returns (uint256)
316 {
317     return vbNEXT.delegatedBalanceAt(_grantors, _delegate, _snapshotTime);
318 }
```

src/spoke/SpokeBridge.sol

```
69 /**
70  * @notice Execute lock position increase on vbNEXT from Spoke
71  * @dev User will need to call with msg.value for fee which should = _fee + ((_f…
72  * @param _additionalAmountToLock The additional amount to lock
73  * @param _expiry expiry time for the lock
74  * @param _gasLimit gas limit to use for Hyperlane
75  * @return _messageId from Hyperlane
76  * @return _feeSpent gas fee spent
77  */
```

src/spoke/SpokeBridge.sol

```
 95 /**
 96  * @notice Execute withdraw on vbNEXT from Spoke
 97  * @param _domain The domain to withdraw from
 98  * @param _gasLimit gas limit to use for Hyperlane
 99  * @return _messageId from Hyperlane
100  * @return _feeSpent gas fee spent
101  */
```

src/spoke/SpokeBridge.sol

```
112 /**
113  * @notice Execute delegate on vbNEXT from Spoke
114  * @param _delegate The address to delegate to
115  * @param _gasLimit gas limit to use for Hyperlane
116  * @return _messageId from Hyperlane
117  * @return _feeSpent gas fee spent
118  */
```

src/spoke/SpokeBridge.sol

```
129 /**
130  * @notice Execute exit early on vbNEXT from Spoke
131  * @param _amountToUnlock The amount to unlock
132  * @param _domain The domain to unlock from
133  * @param _gasLimit gas limit to use for Hyperlane
134  * @return _messageId from Hyperlane
135  * @return _feeSpent gas fee spent
136  */
```

Recommendation

We recommend settling on a single name, e.g., `vbCLEAR`, and enforcing the naming convention across the code base and its documentation. This change should also apply to the technical documentation provided to the assessment team at the beginning of the review.

Minor Inefficient Gas Usage In `_increasePosition` When Parameters Are Zero

Acknowledged

The `_increasePosition` function does not validate whether the input parameters `amountToIncrease` and `durationToIncrease` are non-zero. When both parameters are zero (e.g., when called by the `delegate` function), the function unnecessarily removes the existing position and recreates the same position. This results in redundant operations, including recalculating the balance and updating the slope changes, which waste gas.

In such cases, the function could instead directly update the total supply based on the slope change without modifying the position or balance, thereby optimizing gas usage.

src/hub/vbCLEAR.sol

```
268 function _increasePosition(address user, uint128 amountToIncrease, uint128 durat…
269     internal
270     returns (uint128)
271 {
272     VbBalanceLib.LockedPosition memory oldPosition = positionData[user];
273
274     (VbBalanceLib.VbBalance memory newSupply,) = _applySlopeChange();
275
276     if (!_isCurrentlyExpired(oldPosition.expiry)) {
277         // remove old position not yet expired
278         VbBalanceLib.VbBalance memory oldBalance = oldPosition.convertToVbBalanc…
279         newSupply = newSupply.sub(oldBalance);
280         slopeChanges[oldPosition.expiry] -= oldBalance.slope;
281     }
282
283     VbBalanceLib.LockedPosition memory newPosition =
284         VbBalanceLib.LockedPosition(oldPosition.amount + amountToIncrease, oldPo…
285
286     VbBalanceLib.VbBalance memory newBalance = newPosition.convertToVbBalance();
287     // add new position
288     newSupply = newSupply.add(newBalance);
289     slopeChanges[newPosition.expiry] += newBalance.slope;
290
291     _totalSupply = newSupply;
292     positionData[user] = newPosition;
293     userHistory[user].push(newBalance, _delegate);
294     return newBalance.getCurrentValue();
295 }
```

Recommendation

Add a check in `_increasePosition` to validate that `amountToIncrease` and `durationToIncrease` are non-zero. If both parameters are zero, update the supply directly based on the slope change without modifying the position or balance to optimize gas usage and avoid redundant operations.

Minor Lax Address Validation

Fixed

> This finding has been fixed in the `495bab94ed0ffa941fe37bc0c142151723ef8939` commit by adding further address checks in critical places.

Constructors and setters use address data throughout the code base without proper validation. While benign in most cases, there are cases where immutable state variables are set, and wrong data entry can either invalidate a deployment or result in a loss of funds.

Examples

`hubWETH` is `immutable` and thus should be validated more strictly:

src/spoke/SpokeBridge.sol

```
16 constructor(
17     uint32 _everclearId,
18     address _clear,
19     address _xClear,
20     address _lockbox,
21     address _gateway,
22     address _owner,
23     address _hubCLEAR,
24     address _hubWETH
25 ) Bridge(_everclearId, _gateway, _xClear, _clear, _lockbox, _owner) {
26     gateway = IGateway(_gateway);
27     hubCLEAR = _hubCLEAR;
28     hubWETH = _hubWETH;
29 }
```

As opposed to `treasury`, `bridge` is not sufficiently validated in its constructor as well as its setter:

src/hub/vbCLEAR.sol

```
50 constructor(address _xCLEAR, address _treasury, address _bridge) Ownable(msg.sen…
51     if (_xCLEAR == address(0)) revert ZeroAddress();
52     if (_treasury == address(0)) revert ZeroAddress();
53
54     xCLEAR = IERC20(_xCLEAR);
55     treasury = _treasury;
56     bridge = _bridge;
57     lastSlopeChangeAppliedAt = _getCurrentWeekStart();
58 }
```

src/hub/vbCLEAR.sol

```
65 function updateBridge(address _newBridge) external onlyOwner {
66     address oldBridge = bridge;
67     bridge = _newBridge;
```

```
68     emit BridgeUpdated(oldBridge, _newBridge);
69 }
```

This also applies to the `bridge` reference inside the `RewardDistributor` and `HubGauge` contracts:

src/hub/RewardDistributor.sol

```
81 function updateBridge(address _newBridge) external onlyOwner {
82     address oldBridge = bridge;
83     bridge = _newBridge;
84     emit BridgeUpdated(oldBridge, _newBridge);
85 }
```

src/hub/HubGauge.sol

```
64 constructor(string memory _name, string memory _version, address _vbNEXT, uint25…
65     Ownable(msg.sender)
66     EIP712(_name, _version)
67 {
68     if (_vbNEXT == address(0)) revert ZeroAddress();
69     if (_genesisEpoch == 0) revert ZeroValue();
70
71     vbNEXT = IVbCLEAR(_vbNEXT);
72     genesisEpoch = _genesisEpoch;
73     bridge = _bridge;
74 }
```

src/hub/HubGauge.sol

```
81 function updateBridge(address _newBridge) external onlyOwner {
82     address _oldBridge = bridge;
83     bridge = _newBridge;
84     emit BridgeUpdated(_oldBridge, _newBridge);
85 }
```

Recommendation

For critical addresses such as references to other system components, especially `immutable` ones, we recommend instating zero address checks to prevent faulty updates and potential loss of funds.

Minor Inconsistency In Position Expiration Conditions in `VbBalanceLib`

Fixed

> This finding has been addressed in the following commit:
> `a0b06a22788c9bb1d87d65d88c4169f326f37c7c` by making both expiry checks exclusive.

The `vbBalanceLib` library contains an inconsistency between the `getValueAt` and `isExpired` functions when determining the expiration of a position. Specifically, `getValueAt` considers a position expired if `a.slope * t > a.bias`, while `isExpired` uses the condition `a.slope * uint128(block.timestamp) >= a.bias` (note the strict vs. loose equality condition). In the current codebase, this should not cause problems as `getValuatAt` will return 0 for `a.slope * t == a.bias`. However, it would be preferable to align both boundary conditions for clarity and consistency.

Examples

src/libs/vbBalanceLib.sol

```
51 function isExpired(VbBalance memory a) internal view returns (bool) {
52     return a.slope * uint128(block.timestamp) >= a.bias;
53 }
```

src/libs/vbBalanceLib.sol

```
60 function getValueAt(VbBalance memory a, uint128 t) internal pure returns (uint12…
61     if (a.slope * t > a.bias) {
62         return 0;
63     }
```

Recommendation

One should align the conditions in both functions to ensure consistency.

Minor Lax Validation On `_claims` In `HubBridge._claimRewards`

Fixed

> This finding has been fixed in the `e51aa2eebebffc6305c17ada7b5aa183ffdecfb9` commit on the `audit/creed-reward-array-check` branch by instating the recommended length check and adding a corresponding test case.

The `_claimRewards` function currently assumes that _claims will contain either exactly one or two entries. However, there needs to be a check in place to enforce this requirement. If `_claims` is not in the expected format and has a length other than 1 or 2, the function may behave differently than intended.

src/hub/HubBridge.sol

```
262 function _claimRewards(address _sender, IRewardDistributor.Claim[] memory _claim…
263     bool _tokenOneIsWETH = _claims[0].token == address(WETH);
264
265     rewardDistributor.claim(_claims);
266
267     if (_claims.length == 2) {
268         _tokenOneIsWETH
269             ? _bridgeEth(_sender, _claims[0].amount)
270             : _bridgeOutAndMessage(_sender, _claims[0].amount, _domain);
271         _tokenOneIsWETH
272             ? _bridgeOutAndMessage(_sender, _claims[1].amount, _domain)
273             : _bridgeEth(_sender, _claims[1].amount);
274     } else {
275         _tokenOneIsWETH
276             ? _bridgeEth(_sender, _claims[0].amount)
277             : _bridgeOutAndMessage(_sender, _claims[0].amount, _domain);
278     }
279 }
```

Recommendation

Add a check to ensure that the length of `_claims` is either 1 or 2. If the length does not meet this requirement, the function should revert. This will prevent unintended behavior.

Minor Redundant Initializations In SpokeBridge

Fixed

> This finding has been addressed on the `audit/creed-redundant-logic` branch with the following commit: `a1fc432de5b4397d2c315386d0c0842c940d09bd`.

The `SpokeBridge` contract's constructor initializes its `Bridge` base contract and then sets the `gateway` storage variable. However, the `Bridge` contract has already initialized the gateway.

src/spoke/SpokeBridge.sol

```
16 constructor(
17     uint32 _everclearId,
18     address _clear,
19     address _xClear,
20     address _lockbox,
21     address _gateway,
22     address _owner,
23     address _hubCLEAR,
24     address _hubWETH
25 ) Bridge(_everclearId, _gateway, _xClear, _clear, _lockbox, _owner) {
26     gateway = IGateway(_gateway);
27     hubCLEAR = _hubCLEAR;
28     hubWETH = _hubWETH;
29 }
```

src/common/Bridge.sol

```
30 constructor(
31     uint32 _everclearId,
32     address _gateway,
33     address _xCLEAR,
34     address _CLEAR,
35     address _LOCKBOX,
36     address _owner
37 ) Ownable(_owner) {
38     EVERCLEAR_ID = _everclearId;
39     gateway = IGateway(_gateway);
40     xCLEAR = IXERC20(_xCLEAR);
41     CLEAR = IERC20(_CLEAR);
42     LOCKBOX = IXERC20Lockbox(_LOCKBOX);
43 }
```

Recommendation

The `gateway` initialization on the `SpokeBridge` contract can be safely removed. Furthermore, the constructor parameter names should be clarified as considerable overlap exists.

None Unused Code

Fixed

> This finding has been addressed on the `audit/creed-redundant-logic` branch with the following commits:
>
> - `a1fc432de5b4397d2c315386d0c0842c940d09bd`
> - `a1c406be2a45df41c18774d531d6b5267028d579`
> - `ce5037876709f0d77b124046c4969929b17098cd`

There are several modifier definitions and redundant code areas that are unused. These occurrences can be safely removed.

Examples

Return of shadowed `owner` state variable, overridden by explicit `return` in the function body:

src/hub/HubGauge.sol

```
289 function _getSigner(bytes32 _digest, bytes calldata _signature) internal pure re…
290     return ECDSA.recover(_digest, _signature);
291 }
```

`validAddress` modifier:

src/hub/HubGauge.sol

```
53 modifier validAddress(address _address) {
54     if (_address == address(0)) revert ZeroAddress();
55     _;
56 }
```

`_isExpired`:

src/common/TimeKeeper.sol

```
15 function _isExpired(uint256 expiry, uint256 blockTime) internal pure returns (bo…
16     return (expiry <= blockTime);
17 }
```

None Unchecked ERC20 Transfer And Approval

Acknowledged

> The client mentioned: "On HubBridge transfer is executed after minting XERC20 which has a success check so this should never revert as balance is always sufficient. We're assuming xCLEAR will be correctly configured and is immutable."

Several calls to tokens' `transfer` and `approve` functions throughout the code base have unchecked return values.

src/hub/HubBridge.sol

```
145 if (_success) IERC20(address(xCLEAR)).transfer(_hubError.user, _hubError.amount);
```

src/hub/HubBridge.sol

```
219 if (_success) IERC20(address(xCLEAR)).transfer(_receiver, _amount);
```

src/hub/HubBridge.sol

```
284 IERC20(address(xCLEAR)).approve(address(vbCLEAR), _amount);
```

src/common/Bridge.sol

```
86 CLEAR.approve(address(LOCKBOX), _amount);
```

Recommendation

We recommend implementing checks for the transfer and approval function return values and reverting the transition on failure. The existing retry functionality should also be used when appropriate.

Appendix A

VbBalanceLibFuzzTest Fuzzing Harness

```
pragma solidity ^0.8.20;

import "forge-std/Test.sol";
import "../../src/libs/VbBalanceLib.sol";

contract VbBalanceLibFuzzTest is Test {
    using VbBalanceLib for VbBalanceLib.VbBalance;
    using VbBalanceLib for VbBalanceLib.LockedPosition;

    /// @notice Test the invariant that `sub(add(x, y), y) == x`
    function testFuzz_AddSubInvariant(
        uint128 bias1,
        uint128 slope1,
        uint128 bias2,
        uint128 slope2
    ) public {
        vm.assume(
            bias1 <= type(uint64).max &&
                slope1 <= type(uint64).max &&
                bias2 <= type(uint64).max &&
                slope2 <= type(uint64).max
        ); // prevents overflow
        VbBalanceLib.VbBalance memory x = VbBalanceLib.VbBalance(bias1, slope1);
        VbBalanceLib.VbBalance memory y = VbBalanceLib.VbBalance(bias2, slope2);

        VbBalanceLib.VbBalance memory sum = x.add(y);
        VbBalanceLib.VbBalance memory result = sum.sub(y);

        // Assert that the result matches the original x
        assertEq(result.bias, x.bias, "Bias mismatch in Add-Sub invariant");
        assertEq(result.slope, x.slope, "Slope mismatch in Add-Sub invariant");
    }

    /// @notice Test the invariant that `add(sub(x, y), y) == x` when y <= x
    function testFuzz_SubAddInvariant(
        uint128 bias1,
        uint128 slope1,
        uint128 bias2,
        uint128 slope2
    ) public {
        vm.assume(bias1 >= bias2 && slope1 >= slope2); // Ensure no underflow
        VbBalanceLib.VbBalance memory x = VbBalanceLib.VbBalance(bias1, slope1);
        VbBalanceLib.VbBalance memory y = VbBalanceLib.VbBalance(bias2, slope2);

        VbBalanceLib.VbBalance memory difference = x.sub(y);
        VbBalanceLib.VbBalance memory result = difference.add(y);

        assertEq(result.bias, x.bias, "Bias mismatch in Sub-Add invariant");
        assertEq(result.slope, x.slope, "Slope mismatch in Sub-Add invariant");
    }

    /// @notice Test the invariant that `isExpired()` is consistent with `getCurrentValue()`
    function testFuzz_ExpiredInvariant(uint128 bias, uint128 slope) public {
        VbBalanceLib.VbBalance memory x = VbBalanceLib.VbBalance(bias, slope);
```

```
        bool expired = x.isExpired();
        uint128 currentValue = x.getCurrentValue();

        // If expired, current value should be 0
        if (expired) {
            assertEq(
                currentValue,
                0,
                "Current value should be 0 for expired balance"
            );
        }

        if (x.getCurrentValue() == 0) {
            assertTrue(
                expired,
                "Position shoud be expired for current value == 0"
            );
        }
    }

    /// @notice Test the invariant that `getValueAt(t) == bias - slope * t` for valid t
    function testFuzz_GetValueAtInvariant(
        uint128 bias,
        uint128 slope,
        uint32 t
    ) public {
        VbBalanceLib.VbBalance memory x = VbBalanceLib.VbBalance(bias, slope);

        uint128 value = x.getValueAt(t);
        if (uint256(slope) * t >= bias) {
            assertEq(
                value,
                0,
                "Mismatch in getValueAt invariant - value not 0"
            );
        } else {
            assertEq(
                value,
                bias - slope * t,
                "Mismatch in getValueAt invariant"
            );
        }
    }

    /// @notice Test the invariant that `convertToVbBalance()` properties are maintained
    function testFuzz_ConvertToVbBalanceInvariant(
        uint128 amount,
        uint128 expiry
    ) public {
        // Ensure expiry is within valid bounds
        vm.assume(expiry > 0 && expiry <= VbBalanceLib.MAX_LOCK_TIME);

        VbBalanceLib.LockedPosition memory position = VbBalanceLib
            .LockedPosition(amount, expiry);

        VbBalanceLib.VbBalance memory vb = position.convertToVbBalance();

        assertEq(
            vb.slope,
            amount / VbBalanceLib.MAX_LOCK_TIME,
            "Slope mismatch in convertToVbBalance"
        );
        assertEq(
            vb.bias,
```

```
            vb.slope * expiry,
            "Bias mismatch in convertToVbBalance"
        );

        // Test expiry invariants if slope is non-zero
        if (vb.slope > 0) {
            uint128 calculatedExpiry = vb.getExpiry();
            assertEq(
                calculatedExpiry,
                expiry,
                "Expiry mismatch in convertToVbBalance"
            );
        } else {
            // Ensure getExpiry reverts for zero slope
            vm.expectRevert(
                abi.encodeWithSelector(
                    VbBalanceLib.ZeroSlope.selector,
                    vb.bias,
                    vb.slope
                )
            );
            vb.getExpiry();
        }

        // Test bias consistency with getCurrentValue
        if (block.timestamp < expiry) {
            assertEq(
                vb.getCurrentValue(),
                vb.bias - vb.slope * uint128(block.timestamp),
                "Mismatch in getCurrentValue calculation"
            );
        } else {
            assertEq(
                vb.getCurrentValue(),
                0,
                "CurrentValue should be zero for expired balance"
            );
        }
    }

    function testFuzz_ConvertToVbBalanceAndBack(
        uint128 amount,
        uint128 expiry
    ) public {
        // Ensure expiry is within valid bounds
        vm.assume(expiry > 0 && expiry <= VbBalanceLib.MAX_LOCK_TIME);

        VbBalanceLib.LockedPosition memory originalPosition = VbBalanceLib
            .LockedPosition(amount, expiry);

        VbBalanceLib.VbBalance memory vb = originalPosition
            .convertToVbBalance();

        assertEq(
            vb.slope,
            amount / VbBalanceLib.MAX_LOCK_TIME,
            "Slope mismatch in convertToVbBalance"
        );
        assertEq(
            vb.bias,
            vb.slope * expiry,
            "Bias mismatch in convertToVbBalance"
        );
```

```
        if (vb.slope > 0) {
            uint128 reconstructedAmount = vb.slope * VbBalanceLib.MAX_LOCK_TIME;
            uint128 reconstructedExpiry = vb.bias / vb.slope;

            assertApproxEqRel(
                originalPosition.amount,
                reconstructedAmount,
                1 * 10,
                "Amount mismatch between original and reconstructed LockedPosition exceeds toleranc
            );
            assertEq(
                originalPosition.expiry,
                reconstructedExpiry,
                "Expiry mismatch between original and reconstructed LockedPosition"
            );
        } else {
            // Ensure vb.slope == 0 results in a valid edge case
            assertLt(
                originalPosition.amount,
                10 ** 12, //DUST
                "Amount should be 0 when slope is 0"
            );
            assertLt(
                originalPosition.expiry,
                10 ** 12, //DUST
                "Expiry should be 0 when slope is 0"
            );
        }
    }

    /// @notice Test the invariant for expiry calculation
    function testFuzz_GetExpiryInvariant(uint128 bias, uint128 slope) public {
        vm.assume(slope > 0); // Prevent division by zero
        VbBalanceLib.VbBalance memory x = VbBalanceLib.VbBalance(bias, slope);

        uint128 expiry = x.getExpiry();
        assertEq(expiry, bias / slope, "Mismatch in getExpiry calculation");
    }

    /// @notice Test the invariant for expiry calculation using LockedPosition
    function testFuzz_GetExpiryInvariantWithLockedPosition(
        uint128 amount,
        uint128 expiry
    ) public {
        // Ensure expiry is valid and within bounds
        vm.assume(expiry > 0 && expiry <= VbBalanceLib.MAX_LOCK_TIME);
        VbBalanceLib.LockedPosition memory position = VbBalanceLib
            .LockedPosition(amount, expiry);

        // Convert LockedPosition to VbBalance
        VbBalanceLib.VbBalance memory vb = position.convertToVbBalance();

        // If the slope is zero, the function `getExpiry` should revert
        if (vb.slope == 0) {
            vm.expectRevert(
                abi.encodeWithSelector(
                    VbBalanceLib.ZeroSlope.selector,
                    vb.bias,
                    vb.slope
                )
            );
            vb.getExpiry();
        } else {
            // Verify the expiry calculation
```

```
            uint128 calculatedExpiry = vb.getExpiry();
            assertEq(
                calculatedExpiry,
                vb.bias / vb.slope,
                "Mismatch in expiry calculation from LockedPosition"
            );
        }
    }
}
```

FuzzHelpers Utility Contract

This smart contract has been written to account for implementation details of the Diligence Fuzzing service. It affects the `Helpers` base contract only by setting `_deployContracts` as `virtual`. It has been added in the same directory as `Helpers.sol`.

```
// SPDX-License-Identifier: MIT
pragma solidity ^0.8.20;

import "forge-std/Test.sol";
import {IERC20} from "@openzeppelin/contracts/token/ERC20/IERC20.sol";
import {IERC20Errors} from "@openzeppelin/contracts/interfaces/draft-IERC6093.sol";
import {IVbCLEAR} from "../../src/interfaces/IVbCLEAR.sol";
import {Ownable} from "@openzeppelin/contracts/access/Ownable.sol";
import {ERC1967Proxy} from "@openzeppelin/contracts/proxy/ERC1967/ERC1967Proxy.sol";
import {StandardHookMetadata} from "@hyperlane/hooks/libs/StandardHookMetadata.sol";
import {ERC20Mock} from "@openzeppelin/contracts/mocks/token/ERC20Mock.sol";
import {WETH} from "isolmate/tokens/WETH.sol";

import {Constants} from "./Constants.sol";
import {HubGauge} from "../../src/hub/HubGauge.sol";
import {HubBridge} from "../../src/hub/HubBridge.sol";
import {HubGateway} from "../../src/hub/HubGateway.sol";
import {SpokeBridge} from "../../src/spoke/SpokeBridge.sol";
import {SpokeGateway} from "../../src/spoke/SpokeGateway.sol";
import {vbCLEAR} from "../../src/hub/vbCLEAR.sol";
import {VbBalanceLib} from "../../src/libs/vbBalanceLib.sol";
import {RewardDistributor} from "../../src/hub/RewardDistributor.sol";
import {XERC20} from "../../src/mock/xERC20.sol";
import {ArbSys} from "../../src/mock/ArbSys.sol";
import {XERC20Lockbox} from "../../src/mock/xERC20Lockbox.sol";
import {ITimeKeeper} from "../../src/interfaces/ITimeKeeper.sol";
import {IGateway} from "../../src/interfaces/IGateway.sol";
import {IHubGateway} from "../../src/interfaces/IHubGateway.sol";
import {IBridge} from "../../src/interfaces/IBridge.sol";
import "./Helpers.sol";

contract FuzzHelpers is Helpers {
    ////////////////////////////////// Deployment Helper //////////////////////////////////
    /// @dev Deployment to single chain for unit tests
    function _deployContracts() public override {
        // Gas limit configs
        uint256[] memory _domain = new uint256[](2);
        uint256[] memory _gasLimit = new uint256[](2);
        _domain[0] = EVERCLEAR_ID;
        _gasLimit[0] = 20_000_000;
        _domain[1] = ARBITRUM_ID;
        _gasLimit[1] = 10_000_000;

        // Switching to mainnet fork to deploy Hub contracts for bridging
        // hubId = vm.createSelectFork(vm.envString("MAINNET_RPC"));
```

```
        vm.warp(FIXED_TIMESTAMP);
        WETHHub = address(new WETH());
        everclearSys = address(new ArbSys());
        rewardDistributor = new RewardDistributor(TREASURY);
        genesisEpoch = block.timestamp;
        xCLEARHub = new XERC20("xCLEAR", "xCLEAR", HUB_FACTORY);
        NEXT_TOKEN_HUB = address(xCLEARHub);
        _vbCLEAR = new vbCLEAR(NEXT_TOKEN_HUB, TREASURY, address(hubBridge));
        hubGauge = new HubGauge(
            name,
            version,
            address(_vbCLEAR),
            genesisEpoch,
            address(hubBridge)
        );

        /*////////////////////////////////////////////////////////////
                        ORIGIN DOMAIN
        ////////////////////////////////////////////////////////////*/
        // Deploying the contracts for Spoke related to bridging
        // arbitrumId = vm.createSelectFork(vm.envString("ARBITRUM_RPC"));
        vm.warp(FIXED_TIMESTAMP);
        xCLEAR = XERC20(NEXT_TOKEN);

        // NOTE: Gateway left empty and updated below
        spokeBridge = new TestSpokeBridge(
            EVERCLEAR_ID,
            address(0),
            address(xCLEAR),
            address(0),
            address(0),
            SPOKE_OWNER,
            address(xCLEARHub),
            WETHHub
        );
        spokeGatewayImpl = address(new TestSpokeGateway());

        /*////////////////////////////////////////////////////////////
                        HUB DOMAIN
        ////////////////////////////////////////////////////////////*/
        // vm.selectFork(hubId);
        // NOTE: Gateway left empty and updated below
        hubBridge = new TestHubBridge(
            EVERCLEAR_ID,
            address(0),
            address(xCLEARHub),
            address(0),
            HUB_OWNER,
            address(_vbCLEAR),
            address(hubGauge),
            address(rewardDistributor),
            address(WETHHub),
            address(everclearSys)
        );
        hubGatewayImpl = address(new TestHubGateway());
        hubGateway = TestHubGateway(
            address(
                new ERC1967Proxy(
                    hubGatewayImpl,
                    abi.encodeCall(
                        HubGateway.initialize,
                        (
                            HUB_OWNER,
                            MAILBOX,
```

```
                        address(hubBridge),
                        SECURITY_MODULE
                    )
                )
            )
        )
    );
    _vbCLEAR.updateBridge(address(hubBridge));
    hubGauge.updateBridge(address(hubBridge));
    vm.prank(TREASURY);
    rewardDistributor.updateBridge(address(hubBridge));

    /*///////////////////////////////////////////////////////////////
                    ORIGIN DOMAIN
    ///////////////////////////////////////////////////////////////*/
    // Switching to Arbitrum fork to deploy spokeGateway
    // vm.selectFork(arbitrumId);
    spokeGateway = TestSpokeGateway(
        address(
            new ERC1967Proxy(
                spokeGatewayImpl,
                abi.encodeCall(
                    SpokeGateway.initialize,
                    (
                        HUB_OWNER,
                        MAILBOX,
                        address(spokeBridge),
                        SECURITY_MODULE,
                        EVERCLEAR_ID,
                        _addressToBytes32(address(hubGateway))
                    )
                )
            )
        )
    );

    vm.startPrank(SPOKE_OWNER);
    spokeBridge.updateGateway(address(spokeGateway));
    spokeBridge.updateMessageGasLimit(_domain, _gasLimit);
    vm.stopPrank();

    /*///////////////////////////////////////////////////////////////
                    HUB DOMAIN
    ///////////////////////////////////////////////////////////////*/
    // Switching to mainnet to deploy vvNEXT and configure hubGauge
    // vm.selectFork(hubId);
    // Updating the state of hubBridge and gateway
    vm.startPrank(HUB_OWNER);
    hubBridge.updateGateway(address(hubGateway));
    hubBridge.updateMessageGasLimit(_domain, _gasLimit);
    vm.stopPrank();

    vm.prank(address(hubBridge));
    hubGateway.setChainGateway(
        ARBITRUM_ID,
        _addressToBytes32(address(spokeGateway))
    );
  }
}
```

VbCLEARFuzzTest Fuzzing Harness

This fuzzing harness has been made possible by Dimitar Bounov's significant contributions. His work greatly helped the assessment team in speeding up the testing process and supported the audit's thoroughness.

```
// SPDX-License-Identifier: GPL-3.0-or-later
pragma solidity ^0.8.20;

import "forge-std/Test.sol";
import "../utils/FuzzHelpers.sol";

contract VbCLEARFuzzTest is FuzzHelpers {
    uint constant MAX_UINT =
        115792089237316195423570985008687907853269984665640564039457584007913129639935;
    uint constant ALOT = 2 ** 254;

    function setUp() public {
        _deployContracts();
    }

    function test_increasePosition(
        address user,
        uint128 amount,
        uint128 expiryDivWeek
    ) external {
        uint256 time = block.timestamp;
        vm.assume(expiryDivWeek > time / (7 days));
        vm.assume(expiryDivWeek < (time + 730 days) / (7 days));
        vm.assume(expiryDivWeek > (time + 90 days) / (7 days));
        vm.assume(amount > 0);

        address bridge = _vbCLEAR.bridge();

        // First Increase bridge limits
        vm.prank(HUB_FACTORY);
        xCLEARHub.setLimits(bridge, ALOT, ALOT);

        // Next mint tokens to the bridge
        uint BRIDGE_BALANCE = ALOT; // xCLEARHub.mintingCurrentLimitOf(bridge);
        console.log(BRIDGE_BALANCE);
        //vm.assume(amount <= BRIDGE_BALANCE);
        vm.startPrank(bridge);
        xCLEARHub.mint(bridge, BRIDGE_BALANCE);

        // Next bridge approves _vbCLEAR to take some tokens
        xCLEARHub.approve(address(_vbCLEAR), BRIDGE_BALANCE);
        vm.stopPrank();

        // Finally try increasing our position
        (, uint256 oldExpiry) = _vbCLEAR.positionData(user);
        vm.prank(bridge);
        _vbCLEAR.increaseLockPosition(user, amount, expiryDivWeek * (7 days));
        (, uint256 newExpiry) = _vbCLEAR.positionData(user);

        // Check expiriy increased
        assertGe(newExpiry, oldExpiry);
    }
}
```

File Hashes

- src/hub/HubGateway.sol
  - 48c573c7cd195795ede752a897c73168617c6876f97d6f9be96b2ad52b80dc7b
- src/hub/vbCLEAR.sol
  - 02676a553e3a4cefdccf55d3e46cca417767055aeaa0081a859af1b4413a8854
- src/hub/HubGauge.sol
  - 43cea2b27db84d7c7b51eb8c3947cbaa42b4b5617598a472303c6ec75df74fff
- src/hub/HubBridge.sol
  - a8c834bdee1937a6ff035c1a8082b57b9126f1903990e14de558f49a2aa3257d
- src/hub/RewardDistributor.sol
  - 44c102f32e5e409c96be449d498c838a61cffb6d101fe6ae0f20882b63edda3a
- src/common/TimeKeeper.sol
  - 0554e211f01cb1cb927bce0c8fa14af04984d836e973c6e127285f7ead99da41
- src/common/Bridge.sol
  - 72325b3c4bb0514e6ad541ea593644c822455270eb41050650db04cb3b40c8ad
- src/common/vbTokenBase.sol
  - c73bf861452feb6610d8b5ba90006a07ca44845482c8d7d51c4528b97587ab75
- src/common/Gateway.sol
  - 7e158eb206a1871a1a01b7091cf53ebac0b51daad39ba94a77cad2dfa88af85b
- src/libs/Math.sol
  - 1f2e3f37fc99e43757307a42eca0bb6d63a69871c75b65ec68fe35760aaed9f6
- src/libs/vbBalanceLib.sol
  - cf05cdaa8d8df681c0fbb0250f430379f331c27f1c6d18931c13824a0829ec2e
- src/libs/TypeCasts.sol
  - 0d7a112c53b7f75e6eaea7e31f584eb21d2faef707651c5433ceed1a728af26e
- src/spoke/SpokeGateway.sol
  - 3bb8cf25ff2c0f4a4c9b6d3bbf35a582a5e67725089fba3277c806edab233143
- src/spoke/SpokeBridge.sol
  - ef2ea69bbbd38ec7d52b5b2e2031e02e581f20caa39145e83da042596dcba4f2

Disclaimer

Creed (“CD”) typically receives compensation from one or more clients (the “Clients”) for performing the analysis contained in these reports (the “Reports”). The Reports may be distributed through other means, including via CD publications and other distributions.

The Reports are not an endorsement or indictment of any particular project or team, and the Reports do not guarantee the security of any particular project. This Report does not consider, and should not be interpreted as considering or having any bearing on, the potential economics of a token, token sale or any other product, service or other asset. Cryptographic tokens are emergent technologies and carry with them high levels of technical risk and uncertainty. No Report provides any warranty or representation to any Third-Party in any respect, including regarding the bugfree nature of code, the business model or proprietors of any such business model, and the legal compliance of any such business. No third party should rely on the Reports in any way, including for the purpose of making any decisions to buy or sell any token, product, service or other asset. Specifically, for the avoidance of doubt, this Report does not constitute investment advice, is not intended to be relied upon as investment advice, is not an endorsement of this project or team, and it is not a guarantee as to the absolute security of the project. CD owes no duty to any Third-Party by virtue of publishing these Reports.

PURPOSE OF REPORTS The Reports and the analysis described therein are created solely for Clients and published with their consent. The scope of our review is limited to a review of code and only the code we note as being within the scope of our review within this report. Any Solidity code itself presents unique and unquantifiable risks as the Solidity language itself remains under development and is subject to unknown risks and flaws. The review does not extend to the compiler layer, or any other areas beyond specified code that could present security risks. Cryptographic tokens are emergent technologies and carry with them high levels of technical risk and uncertainty. In some instances, we may perform penetration testing or infrastructure assessments depending on the scope of the particular engagement.

CD makes the Reports available to parties other than the Clients (i.e., “third parties”) – on its website. CD hopes that by making these analyses publicly available, it can help the blockchain ecosystem develop technical best practices in this rapidly evolving area of innovation.

LINKS TO OTHER WEB SITES FROM THIS WEB SITE You may, through hypertext or other computer links, gain access to web sites operated by persons other than CD. Such hyperlinks are provided for your reference and convenience only, and are the exclusive responsibility of such web sites' owners. You agree that CD are not responsible for the content or operation of such Web sites, and that CD shall have no liability to you or any other person or entity for the use of third party Web sites. Except as described below, a hyperlink from this web Site to another web site does not imply or mean that CD endorses the content on that Web site or the operator or operations of that site. You are solely responsible for determining the extent to which you may use any content at any other web sites to which you link from the Reports. CD assumes no responsibility for the use of third party software on

the Web Site and shall have no liability whatsoever to any person or entity for the accuracy or completeness of any outcome generated by such software.

TIMELINESS OF CONTENT The content contained in the Reports is current as of the date appearing on the Report and is subject to change without notice. Unless indicated otherwise, by CD.