Everclear Chimera

August 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 Chimera smart contract system.

Dominik Muhs and Valentin Quelquejay conducted the review over five weeks, from July 29, 2024, to August 30, 2024. A total of 50 person-days were spent.

The Chimera system is composed of both on-chain and off-chain components. It leverages Hyperlane as a transport layer to relay messages between the Hub deployed on a Gelato rollup and the Spokes deployed on different chains. This security review solely focuses on Everclear smart contracts, i.e., the Hub and the Spoke and their dependencies. Other components critical to the system's proper function, such as off-chain systems and components related to the transport transport layer, including its configuration, are not in the scope of this review.

The smart contracts are UUPS upgradeable, allowing for future changes. The contract administrators can adjust some parameters, which could affect user transactions and funds, so these changes should be made carefully. It's crucial that the contracts are owned and managed by (a) secure multisig(s) or governance to protect them. The system's security also depends on the configuration and security of the cross-chain transport layer, Hyperlane. Additionally, the system's reliability depends on the off-chain components and their availability.

Overall, no critical or major-severity findings were identified. However, five medium-severity findings were noted, most notably related to a settlement race condition, possible denial-of-service vectors, and potentially corrupted accounting. Lastly, various minor and informational findings were filed to enforce adherence to security best practices, hardening the smart contract system and decreasing overall operational risk.

In the two days following the engagement, a review of the mitigations was performed. All findings were able to be marked as fixed or acknowledged.

Scope and Objectives

Our review focused on the commit hash `5e5af59ba536b978fc9ac995b8460daa049ad8d0`.

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

- Review the correctness of the intent lifecycle and potential opportunities to break out of it.
- 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 protocol fee calculations are correct and cannot be circumvented.
- 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.

Considering the time frame and risk profile, the development team agreed that off-chain components, the Gelato rollup and Hyperlane, are considered out of scope.

Parallel to the engagement with Creed, the Everclear team pursued internal security reviews and identified and addressed some issues. These internal mitigations and optimizations are addressed in several pull requests. Below, we list the PR reviewed by Creed as part of this engagement:

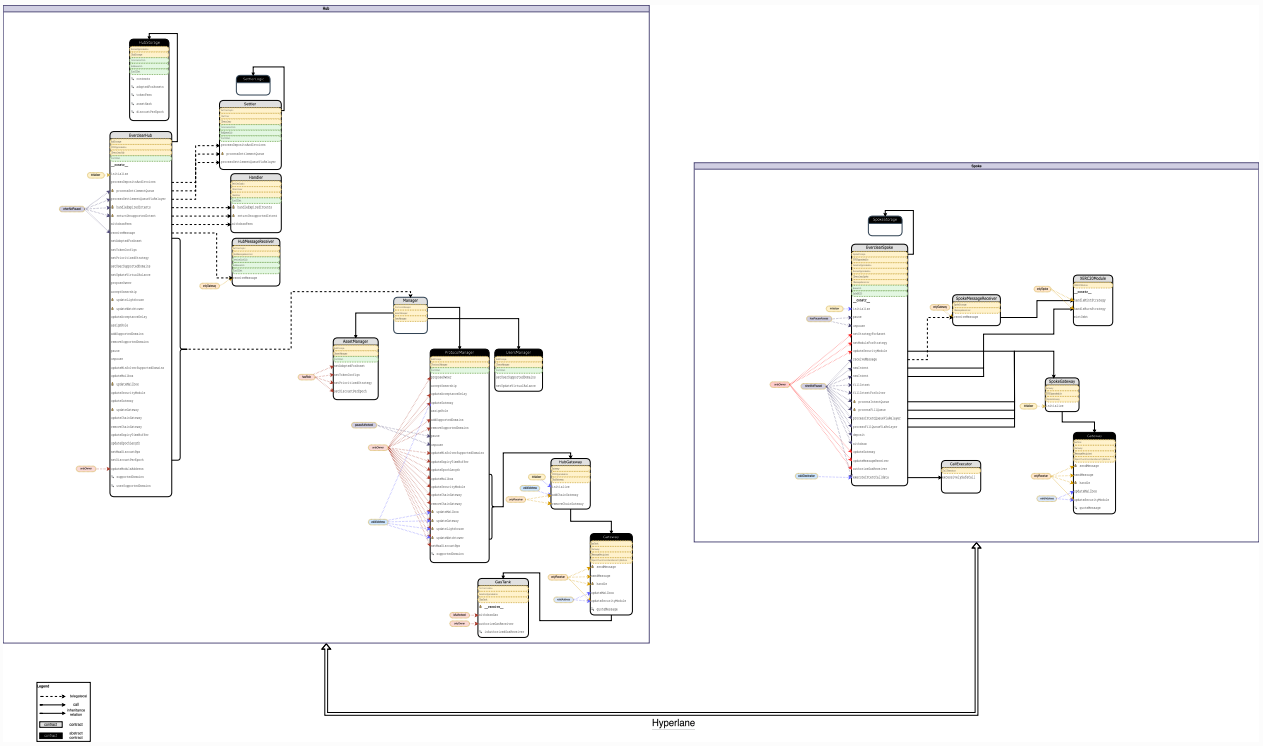
- https://github.com/defi-wonderland/chimera-monorepo/pull/563 (Add limit to `processDepositAndInvoices`)
- https://github.com/defi-wonderland/chimera-monorepo/pull/593 (Zero amount withdrawal check)
- https://github.com/defi-wonderland/chimera-monorepo/pull/582 (Introduce a calldata size limit)

During the mitigations review, the scope was limited to the changes contained in Pull Request #539 until the following commit hash: `04579eac0a6e1c7f7642f4ede7f2cb1cc1f03bf1`.

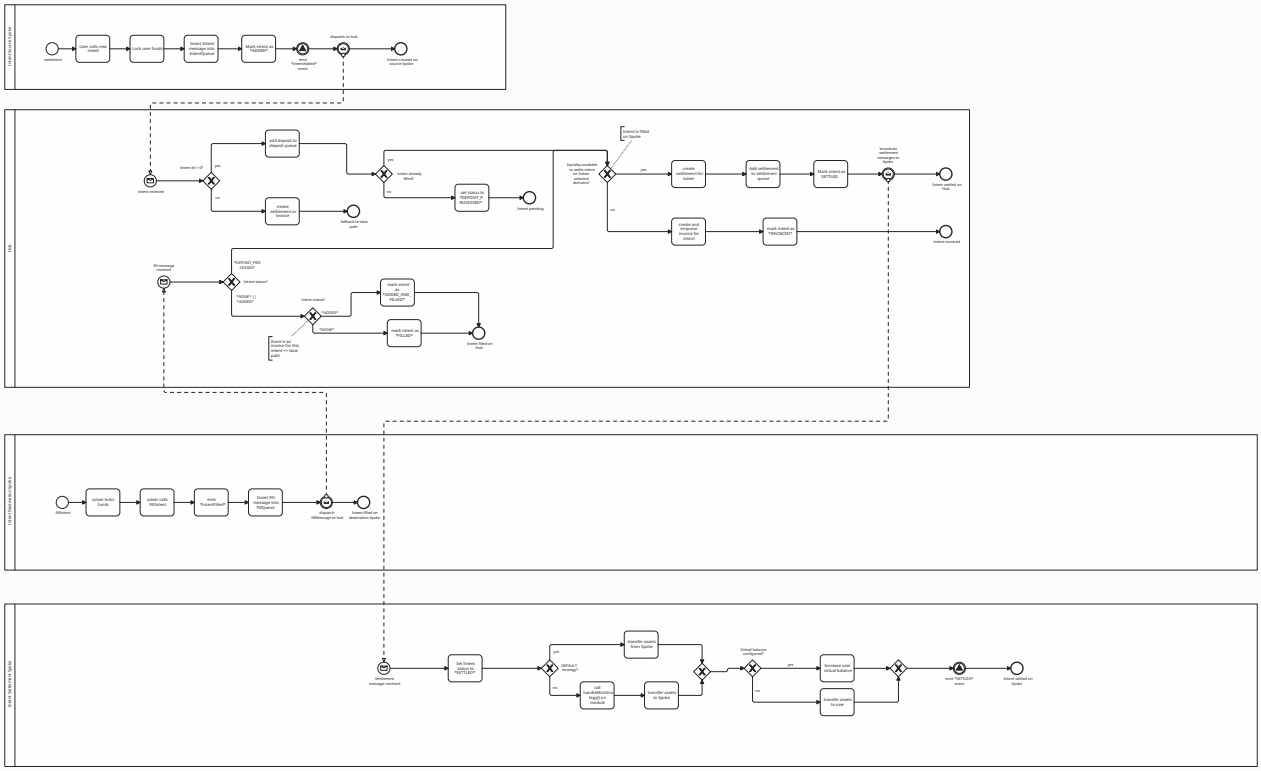
Audit Artifacts

In this section, we provide some artifacts produced during the engagement. Please note that these artifacts might be incomplete or contain incorrect information and should not be used as a reference.

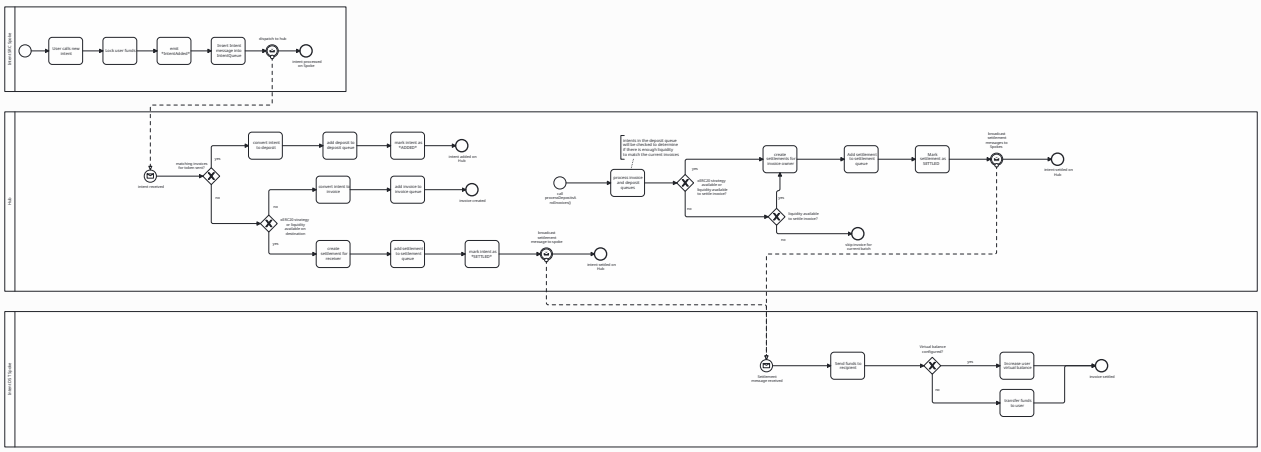
Chimera Architecture:



Intent Lifecycle - fast path:



Intent Lifecycle - slow path:



Additional Considerations

For completeness, below, we list several considerations not directly related to security that emerged during our review.

The *documentation* sometimes needs more consistency, which can make it harder to follow. For instance, terminology varies across different sections, creating confusion. The documentation uses loaded terms like "invoices" and "deposits" in specific contexts without consistently introducing them. The general architecture—particularly the deployment of Spokes across different chains and the hub's role in managing the system—is not very well documented. A clear overview of the system, the components, and their interactions would increase readability. Additionally, introducing a glossary of technical terms in the context of the system would help the reader to read through the documentation.

The system inherently involves cross-chain messaging and execution. When the same spoke smart contract is compiled and deployed on a variety of EVM-compatible chains, security considerations must be taken into account. For compatibility reasons, the smart contracts already correctly use the `bytes32` type for addresses, leaving specific aspects such as gas limits open to configuration. Timestamps are consistently used instead of block numbers, constrained to the hub deployed on a configurable chain. Before deploying a spoke on another chain, the following aspects should be considered to avoid unexpected behavior.

Some chains modify Ethereum opcodes or introduce new ones, impacting contract functionality. For example, Optimism modifies opcodes such as `block.coinbase`, `block.difficulty`, `block.basefee`, and `tx.origin` specifically for L1 to L2 transactions, and it introduces a new opcode called `L1BLOCKNUMBER`. Arbitrum also has opcode variations, altering `blockhash(x)`, `block.coinbase`, `block.difficulty`, `block.number`, and `msg.sender` in scenarios involving "retryable ticket" transactions.

The `PUSH0` opcode, introduced in Solidity version >=0.8.20, may not be universally supported across all chains. This opcode has been integrated into Arbitrum starting from ArbOS 11 and into Optimism following the Canyon Upgrade. However, many other chains may still lack support for `PUSH0`. Consequently, when deploying the contracts compiled with version 0.8.25, this opcode is used, and a new deployment's version may have to be downgraded to be supported.

On chains like Optimism, address aliasing allows contracts on L1 and L2 to have the same address but different bytecodes. This mechanism can impact the behavior of `tx.origin` and `msg.sender`. Under certain circumstances, trust assumptions and contract interactions may break since the expected behavior of these opcodes may differ depending on the chain and transaction type.

Optimism has edge cases where `tx.origin` can equal `msg.sender` in messages from L1 to L2. This difference in behavior could lead to unintended consequences if the contract logic makes assumptions about the values of `tx.origin` and `msg.sender`, e.g., that the two are always distinct.

Hardcoded contract addresses can vary between chains, leading to potential errors in cross-chain interactions. For example, WETH has different addresses on Ethereum and Polygon. This necessitates a thorough check to ensure the correct addresses are used for external contracts on each target chain, preventing miscommunication or failed transactions.

Contracts deployed on different chains may differ in their upgradability status, with some being immutable and others being upgradeable. For instance, USDT is immutable on Ethereum but upgradeable on Polygon.

Contract behavior can also vary significantly across chains, potentially introducing vulnerabilities. On the Gnosis Chain, for example, specific ERC20 tokens such as USD//C implement post-transfer callbacks (`ERC677BridgeToken.transfer -> callAfterTransfer`), unlike their implementations on other chains. This can create opportunities for reentrancy attacks, underlining the fact that before a token is allowed to be listed on the hub, a thorough review of its counterparts on other chains should be performed.

Precompiled contracts, which provide low-level operations, may have different addresses or behave differently, e.g., across chains like Arbitrum and Optimism.

Findings

Medium Decreasing epoch length might impact system accounting and other components

Fixed

> Fixed via commit 6306b207d6a28092258af8fb5b353192b9b917b8 and 841a571a937f1045b66325483654505fb0668de9

The epoch index is computed as `uint48(block.number / epochLength)`. When the `epochLength` is increased, the epoch index decreases, which can result in already processed epochs being processed multiple times.

This affects the invoices and deposits in the queues processed by the function `processDepositsAndInvoices` in the `Settler` contract. Specifically, when `_epoch` is decreased from its previous value while the `deposits` queue is not empty, deposits will remain in the `depositsAvailableInEpoch` mapping since they have not been utilized or converted to invoices and dequeued. However, the liquidity is already accounted for in the `custodiedAssets` mapping, meaning it can be used to settle other invoices in the function `_findLowestDiscountAndHighestLiquidity`.

packages/contracts/src/contracts/hub/modules/HubMessageReceiver.sol

```
68 if (_strategy == IEverclear.Strategy.DEFAULT) {
69   // update the custodied assets balance
70   custodiedAssets[_inputAssetHash] += _intent.amount;
71 }
```

packages/contracts/src/contracts/hub/modules/Settler.sol

```
302 uint256 _selectedLiquidity;
303 TokenConfig storage tokenConfig = _tokenConfigs[_params.tickerHash];
304 _result.discountBps = _getDiscountBps(_params.tickerHash, _params.epoch, _params…
305
306 for (uint256 _i = 0; _i < _params.domains.length; _i++) {
307   uint32 _domain = _params.domains[_i];
308   uint256 _liquidity = custodiedAssets[tokenConfig.assetHashes[_domain]];
309   (uint256 _amountAfterDiscount, uint256 _amountToBeDiscounted, uint256 _rewards…
310     _getDiscountedAmount(_params.tickerHash, _result.discountBps, _domain, _para…
```

Later, when the epoch where the original deposit was first registered is processed, the matching algorithm will mistakenly treat this deposit as if it still has purchasing power even though its liquidity has already been used. This will affect reward computation and deposit settlement, as the depositor may not receive the appropriate rewards and may only receive a settlement based on the available liquidity long after their liquidity has effectively been used.

Furthermore, this will impact the invoice discount calculations since all invoices in the queue will have a zero discount again. The invoice queue will no longer be chronological, meaning new invoices might have a higher discount than the preceding ones. Changing the epoch number will also influence the emitted events. Depending on the off-chain components' data structures, this might result in unexpected behavior and should be tested thoroughly.

Additionally, note that the functions `setTokenConfigs` and `setLastClosedEpochProcessed` in the contract `AssetManager` allow an external actor to set the `lastClosedEpochsProcessed` for specific tickerHashes manually. These functions must be used cautiously as they directly impact the deposits and invoices queues, affecting the accounting of deposits, rewards, and invoice discounts.

We recommend implementing a mechanism to empty deposit queues before the epoch number is modified. Design-wise, the root cause of the issue is that `_currentEpoch()` is dynamically calculating the epoch index based on the `epochLength` setting. We recommend considering a design where a change to `epochLength` does not influence the calculations on already submitted deposits and their epoch-related data.

Medium Possible DoS attack on the Hub contract via `processDepositsAndInvoices`

Fixed

> The Everclear team also found this issue in their internal review and addressed the issue during this engagement in PR #563

The `processDepositsAndInvoices` function in the `Settler` contract iterates over the invoices and deposit queue in while loops to mach invoices with compatible deposits. The function has no control over the range of processed invoices and deposits. Thus, it will loop over the entire range of deposits and invoices for a given ticker.

On cheap chains, an attacker might create many tiny deposits and a bunch of invoices to fill the deposit and invoice queues on the hub so that the `processDepositsAndInvoices` function runs into an out-of-gas error. This will effectively DoS the invoice and deposit matching process for the given ticker hash. Furthermore, there are no ways to manually empty those queues.

packages/contracts/src/contracts/hub/modules/Settler.sol

```
42 bytes32 _invoiceId = _invoiceList.head;
43 bytes32 _previousInvoiceId = 0;
44 while (_invoiceId != 0) {
45   Invoice memory _invoice = _invoiceList.at(_invoiceId).invoice;
46   bytes32 _nextId;
47   _nextId = _invoiceList.at(_invoiceId).next;
48   if (_processInvoice(_epoch, _tickerHash, _invoice)) {
49     // remove the invoice from the list if it was settled
```

packages/contracts/src/contracts/hub/modules/Settler.sol

```
135 Deposit memory _deposit = _depositQueue.at(_index);
136 while (_remainingAmount > 0 && _deposit.intentId != 0) {
137   uint256 _nominator = _deposit.purchasePower > _remainingAmount ? _remainingAmo…
138   uint256 _depositRewards = (_nominator * _domainResult.discountBps) / Constants…
139
140   // tracking remaining rewards for debugging purposes
141   _remainingRewards -= _depositRewards;
```

We recommend upgrading the `processDepositsAndInvoices` function to allow the size of processed invoice and deposit batches to be tweaked, allowing large batches of deposits and invoice to be processed in smaller parts.

Medium `msg.value` in a loop might lead to fund loss or reverting transactions

Fixed

> Fixed by commit 3d41e2c810bdf0fa3a2642ec94fd251cbf3362b1

The `_propagateToDomains` function in the ProtocolManager contract takes an array of domains, _domains, and a _message as arguments. The function iterates over the domains and calls sendMessage on the Hub gateway for each domain to propagate the _message to the different domains via Hyperlane. The function provides `msg.value` as the "value" argument when calling the sendMessage function. However, it is important to note that the `msg.value` is an invariant for the transaction. When funds are sent, the value already sent is not deducted from the transaction's `msg.value` variable. This can lead to two problems:

1. If the Manager contract holds funds, assuming the user sends `x` ETH to the payable function invoking `_propagateToDomains`, the contract will effectively send out `x * len(_domains)` ETH, using funds stored in the contract instead of the funds sent by the user.
2. If there are insufficient funds in the contract, the transaction will revert, and the operation will not be broadcast to the spokes.

packages/contracts/src/contracts/hub/modules/managers/ProtocolManager.sol

```
251 function _propagateToDomains(
252   bytes memory _message,
253   uint32[] memory _domains
254 ) internal returns (bytes32[] memory _messageIds) {
255   _messageIds = new bytes32[](_domains.length);
256   for (uint256 _i = 0; _i < _domains.length; _i++) {
257     (_messageIds[_i],) =
258       IHubGateway(hubGateway).sendMessage{value: msg.value}(_domains[_i], _messa…
259   }
260 }
```

We recommend not sending values in a loop or parameterizing the values of the sub-calls. We also recommend adjusting the accounting so that the value of each call can be set, and the sum of the values in the sub-calls equals `msg.value`.

Medium Race condition could lead to stolen funds

Acknowledged

> Addressed by commit 6f8202967c198732007d6fca96bd960c3665efb9. Additionally, the client mentioned "this will be documented and properly configured once we have a better grasp of the messaging delay across all domains".

The protocol exhibits a race condition that could lead to funds being lost. This race condition can be prevented by adequately configuring parameters with safety buffers according to the system's communication latency, which Hyperlane partly dictates.

1. Eve creates an intent on the source Spoke via the `_newIntent` function. She sets the intent's TTL to a non-zero value and the destination domain to `dst`. The source Spoke sets the intent's status to `IntentStatus.ADDED`.

packages/contracts/src/contracts/intent/EverclearSpoke.sol

```
417    status[_intentId] = IntentStatus.ADDED;
```

2. Hyperlane relays intent queue from source Spoke to Everclear hub. 3. The hub processes the intent queue via the `_processIntents` function. The intent is seen for the first time on the hub. Consequently, the intent's status is set to `IntentStatus.DEPOSIT_PROCESSED` on the hub because, at this point, `intent.ttl != 0` and `intent.solver == 0`.

packages/contracts/src/contracts/hub/modules/HubMessageReceiver.sol

```
80    if (_intent.ttl == 0) {
81      // slow path, no solvers
82      _createSettlementOrInvoice({_intentId: _intentId, _tickerHash: _tickerHash, …
83    } else {
84      bytes32 _solver = _intentContext.solver;
85      // xcall
86      if (_solver == 0) {
87        // intent not filled yet
88        // when deposit is not created is considered processed
89        _intentContext.status = IntentStatus.DEPOSIT_PROCESSED;
90      } else {
91        // fast path, intent filled, settle solver
92        _createSettlementOrInvoice(_intentId, _tickerHash, _solver);
93      }
94    }
95  } else {
```

4. An honest solver `s` fills the intent on `dst` a few seconds/minutes before TTL expires, sending funds to Eve. At this point, `block.timestamp <= _intent.timestamp + _intent.ttl` and `status[_intentId] == IntentStatus.NONE` on the original spoke. The intent status is set to `IntentStatus.Filled` on the destination spoke.

packages/contracts/src/contracts/intent/EverclearSpoke.sol

```
428 function _fillIntent(
429   Intent calldata _intent,
430   address _solver,
431   uint24 _fee
432 ) internal validDestination(_intent) returns (FillMessage memory _fillMessage) {
433   bytes32 _intentId = keccak256(abi.encode(_intent));
434   if (block.timestamp >= _intent.timestamp + _intent.ttl) {
435     revert EverclearSpoke_FillIntent_IntentExpired(_intentId);
436   }
437
438   if (_fee > _intent.maxFee) {
439     revert EverclearSpoke_FillIntent_MaxFeeExceeded(_fee, _intent.maxFee);
440   }
441
442   if (status[_intentId] != IntentStatus.NONE) {
443     revert EverclearSpoke_FillIntent_InvalidStatus(_intentId);
444   }
```

5. The intent expires. Eve calls the function `handleExpiredIntents` with the expired intent. On the hub, the intent status is `IntentStatus.DEPOSIT_PROCESSED`. Assuming `expiryTimeBuffer` is set to a low-enough value, `block.timestamp >= _intent.timestamp + _intent.ttl + expiryTimeBuffer`, so the intent is also considered expired on the hub. Thus, the function calls `_createSettlementOrInvoice` and settles the intent on the destination if liquidity is available. In that case, the intent status is marked as `SETTLED`. Alternatively, the hub creates an invoice and marks the intent as `INVOICED`.

packages/contracts/src/contracts/hub/modules/Handler.sol

```
26 function handleExpiredIntents(bytes32[] calldata _expiredIntentIds) external pay…
27   for (uint256 _i = 0; _i < _expiredIntentIds.length; _i++) {
28     IntentContext storage _intentContext = _contexts[_expiredIntentIds[_i]];
29     Intent memory _intent = _intentContext.intent;
30     bytes32 _intentId = _expiredIntentIds[_i];
31     if (_intentContext.status != IntentStatus.DEPOSIT_PROCESSED) {
32       revert Handler_HandleExpiredIntents_InvalidStatus(_intentId, _intentContex…
33     }
34     if (_intent.ttl == 0) {
35       revert Handler_HandleExpiredIntents_ZeroTTL(_intentId);
36     }
37     // check if the intent is expired
38     if (block.timestamp < _intent.timestamp + _intent.ttl + expiryTimeBuffer) {
39       revert Handler_HandleExpiredIntents_NotExpired(
40         _intentId, block.timestamp, _intent.timestamp + _intent.ttl + expiryTime…
41       );
42     }
43
44     _createSettlementOrInvoice({
45       _intentId: _intentId,
46       _tickerHash: _adoptedForAssets[AssetUtils.getAssetHash(_intent.inputAsset,…
47       _recipient: _intent.receiver
48     });
49   }
```

6. The settlement message is sent from the hub to `dst` and handled by the `_handleSettlement` function. The intent status is `FILLED` on `dst`. Thus, the function proceeds and settles the intent, sending funds to Eve.

packages/contracts/src/contracts/intent/modules/SpokeMessageReceiver.sol

```
86 IntentStatus _intentStatus = status[_message.intentId];
87 // if already settled, ignore (shouldn't happen)
88 if (_intentStatus == IntentStatus.SETTLED || _intentStatus == IntentStatus.SETTL…
89   return;
90 }
91 status[_message.intentId] = IntentStatus.SETTLED;
```

7. The fill queue from `dst` is processed and received on the hub. The function `_processFillMessages` is called. The fill message is ignored since the intent status is either `SETTLED` or `INVOICED`.

packages/contracts/src/contracts/hub/modules/HubMessageReceiver.sol

```
122 if (
123   _previousStatus != IntentStatus.NONE && _previousStatus != IntentStatus.ADDED
124     && _previousStatus != IntentStatus.DEPOSIT_PROCESSED
125 ) {
126   continue;
127 }
```

Eve's intent is settled twice, gaining her funds at the expense of the honest solver's intent, which is never settled, causing him to lose his funds. Note that other ways exist to exploit these race conditions passively without actively calling `handleExpiredIntents` by exploiting latency between

Hub and Spoke communication. Thus, it is crucial to set `expiryTimeBuffer` to a sufficiently high value to ensure that the Fill messages between the spoke and the hub are relayed BEFORE the intent expires on the hub itself.

We recommend configuring the `expiryTimeBuffer` parameter to a safe value and documenting why it is critical to configure this parameter appropriately.

Medium Docker containers are missing user downgrades

Out Of Scope

Several off-chain systems are deployed using Dockerfiles. When using Docker containers, minimizing the privileges the contained software is run under is essential to make lateral movements harder for attackers who gain execution access to the container. In a variety of deployed components, the `USER` directive is missing, which means the software is executed under local `root` privileges:

Poller:

docker/monitor/poller/Dockerfile

```
77 WORKDIR ${LAMBDA_TASK_ROOT}
78 COPY --from=build /tmp/build ${LAMBDA_TASK_ROOT}
79 COPY --from=public.ecr.aws/datadog/lambda-extension:60 /opt/extensions/ /opt/ext…
80
81
82
83 EXPOSE 8080
```

Lighthouse:

docker/lighthouse/Dockerfile

```
73 ARG COMMIT_HASH
74 ENV COMMIT_HASH ${COMMIT_HASH:-unknown}
75
76 # ----- Copy files required at runtime by the app -----
77 WORKDIR ${LAMBDA_TASK_ROOT}
78 COPY --from=build /tmp/build ${LAMBDA_TASK_ROOT}
79 COPY --from=public.ecr.aws/datadog/lambda-extension:60 /opt/extensions/ /opt/ext…
80
81
82
83 EXPOSE 8080
```

Postgres:

docker/db/Dockerfile

```
1 FROM postgres:14
2
3 # Update and install the pg_cron extension
4 RUN apt-get update && apt-get install -y postgresql-14-cron
5
6 RUN echo "shared_preload_libraries='pg_cron'" >> /usr/share/postgresql/postgresq…
```

Cartographer

docker/cartographer/Dockerfile

```
65 FROM node as runtime
66
67 ENV NODE_ENV=production
68
69 ARG COMMIT_HASH
70 ENV COMMIT_HASH ${COMMIT_HASH:-unknown}
71
72 WORKDIR ${LAMBDA_TASK_ROOT}
73 COPY --from=build /tmp/build ${LAMBDA_TASK_ROOT}
74 COPY --from=build /usr/local/bin/dbmate /usr/local/bin/dbmate
75 COPY --from=public.ecr.aws/datadog/lambda-extension:60 /opt/extensions/ /opt/ext…
```

As already done in other Dockerfiles, we recommend downgrading local user privileges.

Minor Intent receiver zero address results in funds getting burned

Acknowledged

When submitting a new intent on a spoke, the `_to` parameter is insufficiently validated, allowing `address(0)` as a possible value. The intent can be filled, but the tokens won't be pushed to the receiver due to the following check:

packages/contracts/src/contracts/intent/EverclearSpoke.sol

```
464 if (_intent.receiver != 0 && _intent.outputAsset != 0 && _amount != 0) {
465   _pushTokens(
466     TypeCasts.bytes32ToAddress(_intent.receiver), TypeCasts.bytes32ToAddress(_in…
467   );
468 }
```

After the hub creates a settlement, it is sent to the spoke, where the `transfer` calls for either the default or the xERC20 strategy are executed. However, if the `_to` parameter is `address(0)`, the funds will become effectively burnt:

packages/contracts/src/contracts/intent/modules/SpokeMessageReceiver.sol

```
108 bytes memory _transferData = abi.encodeWithSignature('transfer(address,uint256)'…
109 (bool _success, bytes memory _res) = _asset.call(_transferData);
```

packages/contracts/src/contracts/intent/modules/XERC20Module.sol

```
40  if (_amount <= _limit) {
41    try IXERC20(_asset).mint(_recipient, _amount) {
42      _success = true;
43    } catch {}
44  }
```

Minor Lack of input sanitization

Fixed

> Fixed by commit d12e54547ea6c7d314112602cbc866ef3682985a

Several functions in the codebase that update configuration parameters do not sanitize their inputs, allowing the setting of invalid configuration parameters.

The function `setTokenConfig` in `AssetManager` enables the `ASSET_MANAGER` to set new token configurations. The asset manager can set new token fees and discounts per epoch/max discount parameters as part of the token configuration. Those parameters are not sanitized, making it possible to set fees and discounts greater than `Common.BPS_DENOMINATOR`, i.e., above 100%.

packages/contracts/src/contracts/hub/modules/managers/AssetManager.sol

```
43 _tokenConfig.discountPerEpoch = _config.discountPerEpoch;
44 _tokenConfig.maxDiscountBps = _config.maxDiscountBps;
45 Fee[] memory _newFees = _config.fees;
```

This also applies to the `setDiscountPerEpoch` function.

packages/contracts/src/contracts/hub/modules/managers/AssetManager.sol

```
82 function setDiscountPerEpoch(bytes32 _tickerHash, uint24 _discountPerEpoch) exte…
83   TokenConfig storage _tokenConfig = _tokenConfigs[_tickerHash];
84   uint24 _oldDiscountPerEpoch = _tokenConfig.discountPerEpoch;
85   _tokenConfig.discountPerEpoch = _discountPerEpoch;
86   emit DiscountPerEpochSet(_tickerHash, _oldDiscountPerEpoch, _discountPerEpoch);
87 }
```

Similarly, the `updateGasConfig` function does not sanitize its parameters.

packages/contracts/src/contracts/hub/modules/managers/ProtocolManager.sol

```
136 function updateGasConfig(GasConfig calldata _newGasConfig) external onlyOwner {
137   GasConfig memory _oldGasConfig = gasConfig;
138   gasConfig = _newGasConfig;
139   emit GasConfigUpdated(_oldGasConfig, _newGasConfig);
140 }
```

We recommend sanitizing parameters to make sure they cannot exceed reasonable thresholds.

Minor Missing constructor initialization

Fixed

> Fixed by commit b54bcb3c0e3104ac935cefe28f9f522d34358f0e

The `HubGateway` contract is missing a constructor calling `_disableInitializers`.

packages/contracts/src/contracts/hub/HubGateway.sol

```
10 contract HubGateway is Gateway, UUPSUpgradeable, IHubGateway {
11   /// @inheritdoc IHubGateway
12   mapping(uint32 _chainId => bytes32 _gateway) public chainGateways;
13
14   /*////////////////////////////////////////////////////////////
15                         GATEWAY FUNCTIONS
16   ////////////////////////////////////////////////////////////*/
17
18   /// @inheritdoc IHubGateway
19   function initialize(address _mailbox, address _receiver, address _interchainSe…
20     _initializeGateway(_mailbox, _receiver, _interchainSecurityModule);
21   }
```

Similarly, the `SpokeGateway` contract does not have a mechanism to disable the initializers either.

packages/contracts/src/contracts/intent/SpokeGateway.sol

```
10 contract SpokeGateway is Gateway, UUPSUpgradeable, ISpokeGateway {
11   uint32 public EVERCLEAR_ID;
12   bytes32 public EVERCLEAR_GATEWAY;
13
14   /*////////////////////////////////////////////////////////////
15                         GATEWAY FUNCTIONS
16   ////////////////////////////////////////////////////////////*/
17
18   /// @inheritdoc ISpokeGateway
19   function initialize(
20     address _mailbox,
21     address _receiver,
22     address _interchainSecurityModule,
23     uint32 _everclearId,
24     bytes32 _hubGateway
25   ) external initializer {
```

In both contracts, we recommend adding a call to `_disableInitializers` in the constructor to prevent unauthorized entities from initializing the implementation.

Minor Incorrect parameters ordering in typehash

Fixed

> Fixed by commit 58e1613cc7e9a44b28881b86583b2021352e8ff8

The parameters ordering defined in the typehash `PROCESS_FILL_QUEUE_VIA_RELAYER_TYPEHASH` does not reflect the actual payload encoding specified in the function `processFillQueueViaRelayer`: `_bufferBPS` and `_nonce` are inverted.

packages/contracts/src/contracts/intent/SpokeStorage.sol

```
28 bytes32 public constant PROCESS_FILL_QUEUE_VIA_RELAYER_TYPEHASH = keccak256(
29   'function processFillQueueViaRelayer(uint32 _domain, uint32 _amount, address _…
30 );
```

packages/contracts/src/contracts/intent/EverclearSpoke.sol

```
268 bytes memory _data =
269   abi.encode(PROCESS_FILL_QUEUE_VIA_RELAYER_TYPEHASH, _domain, _amount, _relayer…
270 _verifySignature(lighthouse, _data, _nonce, _signature);
```

Same applies to `FILL_INTENT_FOR_SOLVER_TYPEHASH`.

packages/contracts/src/contracts/intent/SpokeStorage.sol

```
18 bytes32 public constant FILL_INTENT_FOR_SOLVER_TYPEHASH = keccak256(
19   'function fillIntentForSolver(address _solver, Intent calldata _intent, uint25…
20 );
```

packages/contracts/src/contracts/intent/EverclearSpoke.sol

```
205 bytes memory _data = abi.encode(FILL_INTENT_FOR_SOLVER_TYPEHASH, _intent, _fee, …
```

We recommend fixing the ordering of parameters in the typehash or the encoding.

Minor AssetManager contract should be abstract

Fixed

> Fixed by commit aeba0e8f3469d0ccaf7b695312fae1ea176799f1

The manager contracts in `contracts/hub/modules/managers/` are abstract contracts intended to be merged into a single `Manager` contract. However, unlike the other manager contracts, the `AssetManager` contract is not currently marked as abstract.

packages/contracts/src/contracts/hub/modules/managers/AssetManager.sol

```
12 contract AssetManager is HubStorage, IAssetManager {
13   using Uint32Set for Uint32Set.Set;
14
15   /*////////////////////////////////////////////////////////////////
```

The `AssetManager` contract should be marked as `abstract` since it is not meant to be instantiated directly.

Minor Unused code

Fixed

> Fixed by commit 80934a7a965041aa23280b10c374e59cdec1424e

The codebase contains unused code fragments:

The `GasTank` contract declares an immutable `_OWNER` address. This variable is never initialized (the contract has no constructor) nor referenced anywhere. The contract relies on the `OwnableUpgradeable` library, and the actual contract owner is initialized in the contract `__initializeGasTank` function.

packages/contracts/src/contracts/common/GasTank.sol

```
13    address internal immutable _OWNER;
```

The `_processInvoice` function in the `Settler` contract tracks the `remainingRewards` for debugging purposes, as indicated by a comment in the code. However, the function does not return or store this stack variable anywhere in persistent storage. Thus, it is discarded when the function returns. It should be removed if it is considered redundant.

packages/contracts/src/contracts/hub/modules/Settler.sol

```
131 uint256 _remainingRewards = _domainResult.selectedRewardsForDepositors;
```

packages/contracts/src/contracts/hub/modules/Settler.sol

```
140 // tracking remaining rewards for debugging purposes
141 _remainingRewards -= _depositRewards;
```

We recommend deleting unused code fragments to simplify the codebase, increase readability, and decrease gas costs.

Minor

`_findDestinationXerc20Strategy` might return an xERC20 destination with a depleted mint buffer

Acknowledged

> As per the customer, this is a known limitation of the system. This case is not expected to happen frequently. Users can claim their balances as soon as the buffer replenishes.

The function `_findDestinationXerc20Strategy` does not check if the xERC20 strategy still has available liquidity. Thus, the function might return an xERC20 destination with a depleted mint buffer while other destinations have capacity. This will result in the user failing to claim its tokens immediately, while immediate xERC20 liquidity might be available on other users' configured domains.

packages/contracts/src/contracts/hub/modules/SettlerLogic.sol

```
165   for (uint256 _i = 0; _i < _destinations.length; _i++) {
166     uint32 _destination = _destinations[_i];
167     bytes32 _assetHash = _tokenConfigs[_tickerHash].assetHashes[_destination];
168     if (_adoptedForAssets[_assetHash].strategy == IEverclear.Strategy.XERC20) {
169       _selectedDestination = _destination;
170       break;
171     }
172   }
173 }
```

Minor Missing Events On State-Changing Actions

Fixed

> Fixed by commit 9cad590d831514418b7d08682c02f75ebf7eeca8

Several state-changing functions in the codebase do not emit events.

- In `GasTank`, when a new gas receiver is authorized, no event is emitted

packages/contracts/src/contracts/common/GasTank.sol

```
59 function authorizeGasReceiver(address _receiver, bool _authorized) external only…
60   _authorizedGasReceiver[_receiver] = _authorized;
61 }
```

- In `Settler.processDepositsAndInvoices` after processing-related state changes are applied

packages/contracts/src/contracts/hub/modules/Settler.sol

```
55     _invoiceId = _nextId;
56   }
57
58   // Clean up phase
59   _cleanUpClosedEpochsDeposits(_tickerHash, _epoch);
60 }
```

We recommend that all state-changing functions emit events. This enables tracking and monitoring changes to the contract state, allows off-chain systems to react to the changes, and provides transparency for users.

Minor Inconsistent Use Of Interface And Address Types

Fixed

> Addressed by commit 04579eac0a6e1c7f7642f4ede7f2cb1cc1f03bf1

The codebase contains several instances where the primitive address type is used in place of a contract interface. We recommend using the contract interface type whenever possible, as it allows the Solidity compiler to type-check the calls at compilation. Downcast to the address type only if necessary.

- Gateway mailbox:

packages/contracts/src/contracts/common/Gateway.sol

```
108 _messageId = IMailbox(mailbox).dispatch{value: _fee}(_chainId, _destinationGatew…
```

- Gateway `quoteMessage`

packages/contracts/src/contracts/common/Gateway.sol

```
156 _fee = IMailbox(mailbox).quoteDispatch(_chainId, _gateway, _message, _metadata);
```

- `ProtocolManager` configuration parameters:

packages/contracts/src/contracts/hub/modules/managers/ProtocolManager.sol

```
147 function updateMailbox(address _newMailbox) external onlyOwner {
148   IHubGateway(hubGateway).updateMailbox(_newMailbox);
149 }
150
151 /// @inheritdoc IProtocolManager
152 function updateSecurityModule(address _newSecurityModule) external onlyOwner {
153   IHubGateway(hubGateway).updateSecurityModule(_newSecurityModule);
154 }
155
156 /// @inheritdoc IProtocolManager
157 function updateChainGateway(uint32 _chainId, bytes32 _gateway) external onlyOwne…
158   IHubGateway(hubGateway).addChainGateway(_chainId, _gateway);
159 }
160
161 /// @inheritdoc IProtocolManager
162 function removeChainGateway(uint32 _chainId) external onlyOwner {
163   IHubGateway(hubGateway).removeChainGateway(_chainId);
164 }
```

- `EverclearSpoke` modules:

packages/contracts/src/contracts/intent/EverclearSpoke.sol

```
393 address _module = modules[_strategy];
394 ISettlementModule(_module).handleBurnStrategy(_inputAsset, msg.sender, _amount, …
```

- `SpokeMessageReceiver` modules:

packages/contracts/src/contracts/intent/modules/SpokeMessageReceiver.sol

```
120 address _module = modules[_strategy];
121 address _mintRecipient = _message.updateVirtualBalance ? address(this) : _recipi…
122 bool _success = ISettlementModule(_module).handleMintStrategy(_asset, _mintRecip…
```

None

HubGateway.addChainsGateway can be used to overwrite an existing gateway address

Fixed

> Addressed in commit c5dd012aeb16153d51658171e3a14238e15642f5

The function `HubGateway.addChainsGateway` doesn't revert if a gateway address is already set for a given chain. We recommend renaming the function to reflect that it can be used to update/ overwrite an existing gateway address. Thus, it should be used cautiously as it might impact users' transactions and funds.

packages/contracts/src/contracts/hub/HubGateway.sol

```
38 function addChainGateway(uint32 _chainId, bytes32 _gateway) external onlyReceive…
39   chainGateways[_chainId] = _gateway;
40   emit ChainGatewayAdded(_chainId, _gateway);
41 }
```

None Avoid emitting events on idempotent operations

Acknowledged

We recommend to avoid logging events on idempotent operations. For instance, when calling the `setUpdateVirtualBalance` function with the same `_status` parameter as the one already set.

packages/contracts/src/contracts/hub/modules/managers/UsersManager.sol

```
45 function setUpdateVirtualBalance(bool _status) external {
46   bytes32 _userAddress = TypeCasts.addressToBytes32(msg.sender);
47   updateVirtualBalance[_userAddress] = _status;
48
49   emit IncreaseVirtualBalanceSet(_userAddress, _status);
50 }
```

Or when calling `pause` (resp. `unpause`) if the contract is already paused (resp. unpaused).

packages/contracts/src/contracts/hub/modules/managers/ProtocolManager.sol

```
96         function pause() external pauseAuthorized {
97           paused = true;
98           emit Paused();
99         }
```

packages/contracts/src/contracts/hub/modules/managers/ProtocolManager.sol

```
102       function unpause() external pauseAuthorized {
103         paused = false;
104         emit Unpaused();
105       }
```

File Hashes

- ./contracts/common/QueueLib.sol
  - 3d21869f8a5a64806b33abc91456264b2d3bb00e
- ./contracts/common/Constants.sol
  - 44016e86fc773bc1c15e4ac36858ca27fc5da94c
- ./contracts/common/TypeCasts.sol
  - 45ea72714b63b9adc7c0c7793e5c44ecc4fd566c
- ./contracts/common/MessageLib.sol
  - 0545dee2e0b9e1172e8bee80e4299a15ad6c69b0
- ./contracts/common/AssetUtils.sol
  - 5f8ed1dc578b2f6d9a73955f4cf1d4f255ae83f7
- ./contracts/common/Gateway.sol
  - 0b7009076eacba8e6dea5015e4f2e3934e0365e0
- ./contracts/common/GasTank.sol
  - 2eb5528c5a8509fcad27d8215a2ae2341ac42c5d
- ./contracts/intent/EverclearSpoke.sol
  - fc9a7947e7ecb7e60e7aaa6f6182c4bc07dee8b6
- ./contracts/intent/SpokeGateway.sol
  - 106992fd44b0d879ca0f901d1572f26248ba4631
- ./contracts/intent/lib/Constants.sol
  - f1e1922408c44facfcf1c398b89d269d1636732b
- ./contracts/intent/SpokeStorage.sol
  - bb8555f6ff58d8f327f2b34468ca8940db44ea4a
- ./contracts/intent/modules/XERC20Module.sol
  - 22bc4a46ae63fa1ac55eaf8ca37f4e5d9c678498
- ./contracts/intent/modules/SpokeMessageReceiver.sol
  - 7724cb657caa4a5f9ac687f8ed41c914a6203e34
- ./contracts/intent/CallExecutor.sol
  - dc283eaf3d1932e20372a480833c4f25baf1b207
- ./contracts/hub/lib/HubQueueLib.sol
  - 8605ba896405f1a7ad7d2655ef298e9b2d37d84f
- ./contracts/hub/lib/Uint32Set.sol
  - 40f2677995fe5e321888c71fc7a2fbf6002749d7
- ./contracts/hub/lib/InvoiceListLib.sol
  - 33bb42993f7a9b946236e868f98b8fa06901080f
- ./contracts/hub/HubGateway.sol
  - 1acfb564b5be778da496ac0dd65f8c9a66b26661

- ./contracts/hub/EverclearHub.sol
  - cbc5d17651321b9c6d6f993ae36a9d47da8958cf
- ./contracts/hub/HubStorage.sol
  - 708d7d34d2e7aef364d68125eea97d60e94ab6e9
- ./contracts/hub/modules/Handler.sol
  - 29da3086b252d0d06ff6d9d717134956f012b847
- ./contracts/hub/modules/HubMessageReceiver.sol
  - d00b34eaa2c62a1be5cc72c3b0b3ce1bb5abc0c9
- ./contracts/hub/modules/SettlerLogic.sol
  - 7b7c2bb8f3d8a75ac83862cb63d51ab92ecae6eb
- ./contracts/hub/modules/managers/AssetManager.sol
  - 5bffff2cad9b73d912db8865761234604a10285a
- ./contracts/hub/modules/managers/ProtocolManager.sol
  - 2e3e59266df0f3d1676cc1eb4c539b642239c36e
- ./contracts/hub/modules/managers/UsersManager.sol
  - db8003ac2c81e95964a902dd5345f6b277a76960
- ./contracts/hub/modules/Manager.sol
  - cf7ab5da2dbac25fda6bd6d5a3b9f54555ae10de
- ./contracts/hub/modules/Settler.sol
  - ef34d92ea0cb944804ac48e58d9a3f7b9241db48

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.