Socket

Date	February 2023
Auditors	David Oz, George Kobakhidze

1 Executive Summary

This report presents the results of our engagement with **Socket.tech** to review the smart contracts component of the system.

The review was conducted over three weeks, from **February 13th, 2023** to **March, 3rd 2023**, by **David Oz** and **George Kobakhidze**. A total of 30 person-days were spent.

The assessment was focused on the core parts of the Socket system, including its SocketGateway, Routes, and Controllers architecture. While Socket aims to aggregate a multitude of bridges and decentralized exchanges, this audit focused only on a few specific integrations - Celer bridge, Stargate bridge, and 1inch DEX.

The initial commit hash was `a8d0ad1c280a699d88dc280d9648eacaf215fb41`, which was then switched to `d0841a3e96b54a9d837d2dba471aa0946c3c8e7b` after three days.

2 Scope

Our review focused on the commit hash `d0841a3e96b54a9d837d2dba471aa0946c3c8e7b`. The list of files in scope can be found in the Appendix.

2.1 Objectives

Together with the **Socket.tech** team, we identified the following priorities for our review:

1. Assess if the system is implemented consistently with the intended functionality, and without unintended edge cases, such as:
   - The system is modular and flexible enough to add new integrations and controllers.
   - The system is resilient against users giving infinite approvals.
   - Route id based verifiability of calldata on integrators' end.
   - Only privileged role holders can add new and pause existing routes.
   - Only intended addresses should receive swapped or bridged funds.
2. Identify known vulnerabilities particular to smart contract systems, as outlined in our Smart Contract Best Practices, and the Smart Contract Weakness Classification Registry.

3 Recommendations

3.1 Consider Adding a Task to CI/CD to Verify That Future Delegatee Contracts Are Safe

Description

The system is based on delegating calls to routes and controllers. delegatee contracts should not write to storage, self-destruct, or delegate-call to unknown contracts. In addition, caution is needed when using `msg.value` as mentioned in issue 6.13. We were not able to find any concrete instances of the described issue, however, we do see how these pitfalls may become an issue in future delegatee contracts.

Recommendation

Consider using a package like https://github.com/OpenZeppelin/openzeppelin-upgrades or an equivalent.

3.2 Consider Using a Package Manager Instead of Vendor Code

Description

When you vendor a library, you essentially copy the library code into your project's codebase, which can lead to problems with version control and code management. If the vendor library code changes, you'll need to manually update the code in your project, which can be time-consuming and error-prone. Additionally, if the vendor library has any security vulnerabilities, copying the code into your project can make it more difficult to address those vulnerabilities. Instead, it's generally better to use the Solidity package manager, which allows you to import libraries into your project without copying their code. This way, you can easily update the library code, and any security vulnerabilities can be addressed centrally.

Errors that arise when copying interfaces can have far-reaching consequences, particularly when it comes to functions that are mislabeled as view or pure in the interface but are actually state-changing functions. If such functions are called they have the potential to cause an entire transaction to revert, leading to a potential denial of service.

Examples

src/libraries/LibBytes.sol:L4

```
library LibBytes {
```

src/libraries/Pb.sol:L6

```
library Pb {
```

3.3 Consider Adding Non-Reentrant Modifiers to State-Changing Functions in `SocketGateway`

Description

`SocketGateway` is the main contract for user interaction in the system. The contract is designed to be used by multiple users where the main flow is that a user is depositing funds to the contract and chooses how these funds should be used by external contracts. Currently, we were not able to find any concrete issues that are caused by reentrancies, however, given the fact that the system is planned to be expanded by the use of delegatecalls, it is recommended to add `nonReentrant` modifiers to state changing functions inside `SocketGateway`.

3.4 Use the Same Solidity Version Across Contracts

Description

Most contracts use the same range for Solidity versions with `pragma solidity ^0.8.4`. There are also many that use `pragma solidity >=0.8.0`.

Recommendation

Lock in a specific version of solidity or at least pick a consistent range. This would help avoid any issues and inconsistencies that may arise in deploying the various smart contracts across different Solidity versions.

3.5 Gas Optimizations

Resolution
Remediated as per the client team in SocketDotTech/socket-ll-contracts#154.

Description

The client mentioned that gas was important. Optimizing for gas should **never** come at the cost of security. However, we noticed a few optimizations that could be made.

Examples

`socketGateway` can be replaced with `address(this)`

src/bridges/cbridge/CelerImpl.sol:L320

```
tokenInstance.safeTransferFrom(msg.sender, socketGateway, amount);
```

src/bridges/cbridge/CelerImpl.sol:L412

```
uint256 _initialBalanceTokenOut = socketGateway.balance;
```

src/bridges/cbridge/CelerImpl.sol:L417

```
if (request.receiver != socketGateway) {
```

src/bridges/cbridge/CelerImpl.sol:L430

```
if (socketGateway.balance > _initialBalanceTokenOut) {
```

As discussed with the client, resetting the approval to zero after the swap is implemented to prevent a future USDT approval from reverting in case the previous swap didn't consume the entire allowance (as the USDT contract requires resetting the allowance to zero, before changing it). However, this should not happen in a system that behaves properly. To save gas, we recommend removing this check and implementing a gateway function that allows setting a token allowance to zero. Note that this function does not need to be protected, as it only allows setting the gateway token allowance to zero.

src/swap/oneinch/OneInchImpl.sol:L64

```
token.safeApprove(ONEINCH_AGGREGATOR, 0);
```

src/swap/oneinch/OneInchImpl.sol:L123

```
token.safeApprove(ONEINCH_AGGREGATOR, 0);
```

3.6 Duplicated Code

Description

Duplicate code, or code that is copied and pasted multiple times within a project or across projects, is generally not a good practice in software development. It can lead to several issues, including increased maintenance costs, decreased code readability, and a higher likelihood of introducing bugs into the codebase. When code is duplicated, any changes that need to be made must be replicated across all instances of the code, which can be time-consuming and error-prone. Additionally, duplicated code can make it harder to understand the overall structure of the codebase, as well as make it more difficult to identify and fix issues when they arise.

Examples

- `FeesTakerController` - functions in this contract share similar logic that can be de-duplicated.
- `OneInchImpl` - functions in this contract share similar logic that can be de-duplicated.
- `CelerImpl` - functions in this contract share similar logic that can be de-duplicated.
- `Stargate L1`, `Stargate L2` - the contracts themselves are pretty similar.
- `SwapImplBase`, `BridgeImplBase` - the contracts themselves are pretty similar.

Recommendation

To mitigate these problems, it's often better to refactor duplicated code into reusable functions or classes or to find other ways to modularize the code and reduce redundancy. By doing so, code can be more easily maintained, tested, and extended over time, leading to a more robust and reliable software application.

4 System Overview

The Socket system of contracts aims to provide its users with an easy means to access a multitude of common and relatively interchangeable solutions, such as bridges and swaps. Essentially, the system aggregates access to different systems to just the one `SocketGateway` contract that performs `delegatecall` transactions into `routes`, implementation contracts that hold logic necessary to interact with integrated solutions. This is done by managing contract addresses as routes in the registry maintained in the `SocketGateway` that is able to easily (although with `Owner` administrative power only) add and disable them but never change an existing one. Hence, the name of the system - Socket. The Socket team can create and remove routes but can't change the logic within already registered ones.

The contracts are written with a high emphasis on immutability and gas efficiency, trying to minimize state variables and changes to them as much as possible. Even the deployment of future integrations is designed to be more gas efficient by pre-populating addresses of implementation contracts that can be derived through the `CREATE2` opcode in the `SocketDeployFactory`, as described more below.

As users interact with the system, they should take note of what routes, and therefore the associated bridge or swap solutions, they will be interacting with. The Socket system adds a helpful aggregated wrapper around these destinations, but it does not improve upon their own security. In fact, since the route IDs will always refer to the same address they were assigned to, and the contracts at these addresses have either the same logic as they did in the beginning or are disabled, users and systems integrating Socket could come up with a whitelist of route IDs to ensure they always go through those solutions that they deem to be trusted.

Below you may find an overview diagram of the system's scope for this audit and small descriptions of major components:

4.1 SocketGateway

This is the entry point into the whole Socket system. Both users and Socket team via the `SocketGateway`'s `Owner` will call into this contract to interact with this system. It contains the registry of all route IDs and their associated implementation contracts.

4.2 SocketDeployFactory

In order to be more efficient, the Socket team decided to pre-populate 512 route ID addresses by calculating them via the `CREATE2` opcode. This deployment pattern is done through the `SocketDeployFactory`, that is also able to destroy and disable existing routes.

4.3 BridgeImplBase & SwapImplBase

The two common types of routes, `BridgeImplBase` and `SwapImplBase` are base contracts that implement default functionality for routes, such as `isSocketGatewayOwner` modifiers, token rescue functions, and virtual use-case-specific functions to be re-implemented for bridge and swap routes.

4.4 BaseController

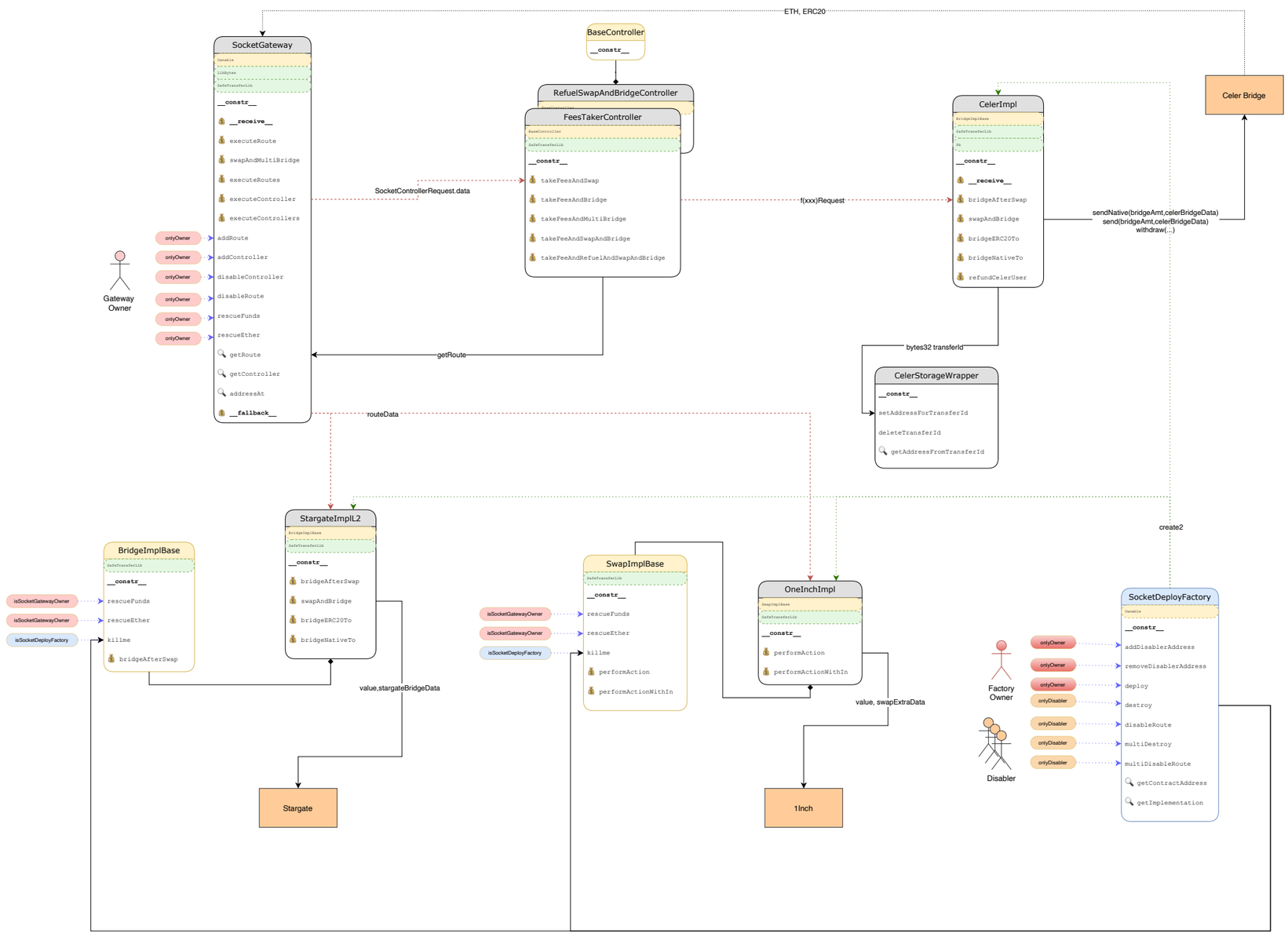
This is the base contract that implements basic functionality required for the controllers. Unlike routes that lead into bridges and swap solutions, controllers may have more complex logic, such as minimal refueling of native tokens for the recipient address and implementing extra fees.

4.5 ***StorageWrapper

Some specific cases, like the Celer bridge, require storage of additional variables. Since routes and controllers don't have storage that is used by the `SocketGateway` due to the usage of `delegatecall`, additional storage wrapper contracts need to be deployed to keep track of extra varaibles.

4.6 Integrated Solution (1inch, Celer, Stargate etc.)

These are the downstream solutions that are aggregated by Socket and perform the requested task for the users, such as swaps and bridge transfers.



5 Security Specification

This section describes, **from a security perspective**, the expected behavior of the system under audit. It is not a substitute for documentation. The purpose of this section is to identify specific security properties that were validated by the audit team.

The primary security concept for the Socket system is that it is an aggregator contract. Socket provides a way for users to interact with many other solutions, but, in its current form, it does not provide any additional security guarantees for the systems it integrates. The security is wholly inherited for bridge transfers and swaps that it performs as the data payloads that users provide are simply passed through to the downstream solutions.

The Socket system itself does have an emphasis on the immutability of the contracts it deploys, however. The contract addresses inserted in its registry can be, at most, disabled, but they can't have their logic be changed to something else. There are a few admin-controlled functions which are mostly isolated to adding new routes and controllers, but there are exceptions where `SocketGateway` `Owner` addresses may have privileged access into the control and data flow of route logic, like the case with the Celer bridge. In the current scope this access is limited to refund and failure recovery cases.

5.1 Actors

The relevant actors are listed below:

- The Socket team
- The integrated systems
- End users & integrating systems

5.2 Trust Model

In any system, it's important to identify what trust is expected/required between various actors. For this audit, we established the following trust model:

Socket team:

In the current system, the Socket team is responsible for several critical components to ensure the system's correct behavior.

First, the team quite simply has administrative abilities over the `SocketGateway` and its routes via the `onlyOwner` modifier. These include management functions like `addRoute`, `addController`, `disableRoute`, and `disableController`, as well as the ability to pull out stuck tokens from the contracts via `rescueEther` and `rescueFunds`. As referred to in one of the filed issues, the ability to add arbitrary logic via `addRoute` and `addController` that will execute via `delegatecall` from `SocketGateway` as `msg.sender` may create edge cases where the `Owner` address can actually have access to user funds and can steal them, like the case with the Celer bridge refunds.

Second, the Socket team also provides and connects to APIs that compile the payloads necessary for correct route execution. These payloads are encoded (although sometimes in a simple manner) as determined by the route's integrated system and are not always easy for users to understand nor necessarily checked against other user-provided inputs, such as swap amounts. This, however, can be dealt with by more advanced users by compiling the necessary payloads themselves, although it is unlikely to be the common use case

As a result, the Socket team itself is trusted to at least set up the appropriate routes and not have its privileged access compromised in certain edge cases for the system to function correctly.

Integrated systems:

The systems that Socket routes to likewise play a crucial role since they are what is being aggregated. They are trusted to continue operating as assumed by their specifications (such as implementing swaps and bridge transfers correctly) and to treat the Socket system as any other smart contract using these solutions. As a result, their security assumptions are inherited in the Socket system as well.

End users

Finally, the end users are the actors that make the system go. They provide the funds that go through the system and hop between different routes. Due to the nature of the solutions that Socket integrates, the end users' interactions with the gateway start and end with a single transaction in the vast majority of the cases, the exceptions being cases like refund scenarios with specific bridges. As a result, the users need not act in any specific way or perform any duties beyond their initial transactions. Similarly, other systems can act as end users themselves and use and integrate the Socket system into their processes. In fact, due to the nature of the Socket route and controller IDs, these systems can assess and compile whitelists of IDs to ensure that only certain DEXs/bridges/logic are executed through this system, so there is some granularity as to how the Socket system can be integrated.

5.3 Security Properties

The following is a non-exhaustive list of security properties that were assessed in this audit:

System immutability

The routes and controllers that are maintained in the `SocketGateway` registry can be disabled but are immutable otherwise. No new code can be added in their place, with the exception being a revert-only option.

Contract balances non-increasing

The `SocketGateway` and its associated routes and controllers are not meant to hold any tokens by design. This, however, can sometimes fail due to refund mechanisms of specific bridges as well as due to incorrectly submitted payloads for swap routes. The system will not benefit or interact with stuck tokens in any way, and there are rescue `onlyOwner` functions that could help retrieve the tokens. Nonetheless, there are cases when these stuck tokens, particularly native chain tokens, could actually be stolen by malicious actors, as referred to in a few filed issues. It is recommended to pay close attention to token balances on the gateway and rescue them swiftly.

6 Findings

Each issue has an assigned severity:

- Minor issues are subjective in nature. They are typically suggestions around best practices or readability. Code maintainers should use their own judgment as to whether to address such issues.
- Medium issues are objective in nature but are not security vulnerabilities. These should be addressed unless there is a clear reason not to.
- Major issues are security vulnerabilities that may not be directly exploitable or may require certain conditions in order to be exploited. All major issues should be addressed.
- Critical issues are directly exploitable security vulnerabilities that need to be fixed.

6.1 Funds Refunded From Celer Bridge Might Be Stolen Major

Resolution

Remediated as per the client team in SocketDotTech/socket-ll-contracts#144 by adding checks to see if the refund is received and equal to the expected amount.

Description

The function `refundCelerUser` from `CelerImpl.sol` allows a user that deposited into the Celer pool on the source chain, to be refunded for tokens that were not bridged to the destination chain. The tokens are reimbursed to the user by calling the `withdraw` method on the Celer pool. This is what the `refundCelerUser` function is doing.

src/bridges/cbridge/CelerImpl.sol:L413-L415

```
if (!router.withdraws(transferId)) {
    router.withdraw(_request, _sigs, _signers, _powers);
}
```

From the point of view of the Celer bridge, the initial depositor of the tokens is the `SocketGateway` . As a consequence, the Celer contract transfers the tokens to be refunded to the gateway. The gateway is then in charge of forwarding the tokens to the initial depositor. To achieve this, it keeps a mapping of unique transfer IDs to depositor addresses. Once a refund is processed, the corresponding address in the mapping is reset to the zero address.

Looking at the `withdraw` function of the Celer pool, we see that for some tokens, it is possible that the reimbursement will not be processed directly, but only after some delay. From the gateway point of view, the reimbursement will be marked as successful, and the address of the original sender corresponding to this transfer ID will be reset to address(0).

```
if (delayThreshold > 0 && wdmsg.amount > delayThreshold) {
     _addDelayedTransfer(wdId, wdmsg.receiver, wdmsg.token, wdmsg. // <--- here
} else {
      _sendToken(wdmsg.receiver, wdmsg.token, wdmsg.
}
```

It is then the responsibility of the user, once the locking delay has passed, to call another function to claim the tokens. Unfortunately, in our case, this means that the funds will be sent back to the gateway contract and not to the original sender. Because the gateway implements `rescueEther` , and `rescueFunds` functions, the admin might be able to send the funds back to the user. However, this requires manual intervention and breaks the trustlessness assumptions of the system. Also, in that case, there is no easy way to trace back the original address of the sender, that corresponds to this refund.

However, there is an additional issue that might allow an attacker to steal some funds from the gateway. Indeed, when claiming the refund, if it is in ETH, the gateway will have some balance when the transaction completes. Any user can then call any

function that consumes the gateway balance, such as the `swapAndBridge` from `CelerImpl`, to steal the refunded ETH. That is possible as the function relies on a user-provided amount as an input, and not on `msg.value`. Additionally, if the refund is an ERC-20, an attacker can steal the funds by calling `bridgeAfterSwap` or `swapAndBridge` from the `Stargate` or `Celer` routes with the right parameters.

src/bridges/cbridge/CelerImpl.sol:L120-L127

```
function bridgeAfterSwap(
    uint256 amount,
    bytes calldata bridgeData
) external payable override {
    CelerBridgeData memory celerBridgeData = abi.decode(
        bridgeData,
        (CelerBridgeData)
    );
```

src/bridges/stargate/l2/Stargate.sol:L183-L186

```
function swapAndBridge(
    uint32 swapId,
    bytes calldata swapData,
    StargateBridgeDataNoToken calldata stargateBridgeData
```

Note that this violates the security assumption: “The contracts are not supposed to hold any funds post-tx execution.”

Recommendation

Make sure that `CelerImpl` supports also the delayed withdrawals functionality and that withdrawal requests are deleted only if the receiver has received the withdrawal in a single transaction.

6.2 Calls Made to Non-Existent/Removed Routes or Controllers Will Not Result in Failure Major

Resolution
Remediated as per the client team in SocketDotTech/socket-ll-contracts#145 by adding a `disabledRouteAddress` contract to be returned for disabled routes instead of a `address(0)`.

Description

This issue was found in commit hash `a8d0ad1c280a699d88dc280d9648eacaf215fb41`.

In the Ethereum Virtual Machine (EVM), `delegatecall` will succeed for calls to externally owned accounts and more specifically to the zero address, which presents a potential security risk. We have identified multiple instances of `delegatecall` being used to invoke smart contract functions.

This, combined with the fact that routes can be removed from the system by the owner of the `SocketGateway` contract using the `disableRoute` function, makes it possible for the user’s funds to be lost in case of an `executeRoute` transaction (for instance) that’s waiting in the mempool is eventually being front-ran by a call to `disableRoute`.

Examples

src/SocketGateway.sol:L95

```
(bool success, bytes memory result) = addressAt(routeId).delegatecall(
```

src/bridges/cbridge/CelerImpl.sol:L208

```
.delegatecall(swapData);
```

src/bridges/stargate/l1/Stargate.sol:L187

```
.delegatecall(swapData);
```

src/bridges/stargate/l2/Stargate.sol:L190

```
.delegatecall(swapData);
```

src/controllers/BaseController.sol:L50

```
.delegatecall(data);
```

Even after the upgrade to commit hash `d0841a3e96b54a9d837d2dba471aa0946c3c8e7b`, the following bug is still present:

To optimize gas usage, the `addressAt` function in `socketGateway` uses a binary search in a hard-coded table to resolve a `routeID` (`routeID <= 512`) to a contract address. This is made possible thanks to the factory using the `CREATE2` pattern. This allows to pre-compute future addresses of contracts before they are deployed. In case the `routeID` is strictly greater than 512, `addressAt` falls back to fetching the address from a state mapping (`routes`).

The new commit hash adds a check to make sure that the call to the `addressAt` function reverts in case a `routeID` is not present in the `routes` mapping. This prevents delegate-calling to non-existent addresses in various places of the code. However, this does not solve the issue for the hard-coded route addresses (i.e., `routeID <= 512`). In that case, the `addressAt` function still returns a valid

route contract address, despite the contract not being deployed yet. This will result in a successful `delegatecall` later in the code and might lead to various side-effects.

src/SocketGateway.sol:L411-L428

```
function addressAt(uint32 routeId) public view returns (address) {
    if (routeId < 513) {
        if (routeId < 257) {
            if (routeId < 129) {
                if (routeId < 65) {
                    if (routeId < 33) {
                        if (routeId < 17) {
                            if (routeId < 9) {
                                if (routeId < 5) {
                                    if (routeId < 3) {
                                        if (routeId == 1) {
                                            return
                                                0x822D4B4e63499a576Ab1cc152B86D1CFFf794F4f;
                                        } else {
                                            return
                                                0x822D4B4e63499a576Ab1cc152B86D1CFFf794F4f;
                                        }
                                    } else {
```

src/SocketGateway.sol:L2971-L2972

```
if (routes[routeId] == address(0)) revert ZeroAddressNotAllowed();
return routes[routeId];
```

Recommendation

Consider adding a check to validate that the callee of a `delegatecall` is indeed a contract, you may refer to the Address library by OZ.

6.3 Owner Can Add Arbitrary Code to Be Executed From the SocketGateway Contract Medium

Resolution

The client team has responded with the following note:

> Noted, we will setup tests and rigorous processes around adding new routes.

Description

The Socket system is managed by the `SocketGateway` contract that maintains all routes and controller addresses within its state. There, the address with the `Owner` role of the `SocketGateway` contract can add new routes and controllers that would have a `delegatecall()` executed upon them from the `SocketGateway` so user transactions can go through the logic required for the bridge, swap, or any other solution integrated with Socket. These routes and controllers would then have arbitrary code that is entirely up to the `Owner`, though users are not required to go through any specific routes and can decide which routes to pick.

Since these routes are called via `delegatecall()`, they don't hold any storage variables that would be used in the Socket systems. However, as Socket aggregates more solutions, unexpected complexities may arise that could require storing and accessing variables through additional contracts. Those contracts would be access control protected to only have the `SocketGateway` contract have the privileges to modify its variables.

This together with the `Owner` of the `SocketGateway` being able to add routes with arbitrary code creates an attack vector where a compromised address with `Owner` privileges may add a route that would contain code that exploits the special privileges assigned to the `SocketGateway` contract for their benefit.

For example, the Celer bridge needs extra logic to account for its refund mechanism, so there is an additional `CelerStorageWrapper` contract that maintains a mapping between individual bridge transfer transactions and their associated `msg.sender`:

src/bridges/cbridge/CelerImpl.sol:L145

```
celerStorageWrapper.setAddressForTransferId(transferId, msg.sender);
```

src/bridges/cbridge/CelerStorageWrapper.sol:L6-L12

```
/**
 * @title CelerStorageWrapper
 * @notice handle storageMappings used while bridging ERC20 and native on CelerBridge
 * @dev all functions ehich mutate the storage are restricted to Owner of SocketGateway
 * @author Socket dot tech.
 */
contract CelerStorageWrapper {
```

Consequently, this contract has access-protected functions that may only be called by the SocketGateway to set and delete the transfer IDs:

src/bridges/cbridge/CelerStorageWrapper.sol:L32

```
function setAddressForTransferId(
```

src/bridges/cbridge/CelerStorageWrapper.sol:L52

```
function deleteTransferId(bytes32 transferId) external {
```

A compromised `Owner` of SocketGateway could then create a route that calls into the `CelerStorageWrapper` contract and updates the transfer IDs associated addresses to be under their control via `deleteTransferId()` and `setAddressForTransferId()` functions. This could create a significant drain of user funds, though, it depends on a compromised privileged `Owner` address.

Recommendation

Although it may indeed be unlikely, for aggregating solutions it is especially important to try and minimize compromised access issues. As future solutions require more complexity, consider architecting their integrations in such a way that they require as few administrative and SocketGateway-initiated transactions as possible. Through conversations with the Socket team, it appears that solutions such as timelocks on adding new routes are being considered as well, which would help catch the problem before it appears as well.

6.4 Dependency on Third-Party APIs to Create the Right Payload Medium

Resolution

The client team has responded with the following note:

> We offset this risk by following 2 approaches - verifying oneinch calldata on our api before making full calldata for SocketGateway and making verifier contracts/libs that integrators can use to verify our calldata on their side before making actual transaction.

Description

The Socket system of routes and controllers integrates swaps, bridges, and potentially other solutions that are vastly different from each other. The function arguments that are required to execute them may often seem like a black box of a payload for a typical end user. In fact, even when users explicitly provide a destination `token` with an associated `amount` for a swap, these arguments themselves might not even be fully (or at all) used in the route itself. Instead, often the routes and controllers accept a `bytes` payload that contains all the necessary data for its action. These data payloads are generated off-chain, often via centralized APIs provided by the integrated systems themselves, which is understandable in isolation as they have to be generated somewhere at some point. However, the provided bytes do not get checked for their correctness or matching with the other arguments that the user explicitly provided. Even the events that get emitted refer to the individual arguments of functions as opposed to what actually was being used to execute the logic.

For example, the implementation route for the 1inch swaps explicitly asks the user to provide `fromToken`, `toToken`, `amount`, and `receiverAddress`, however only `fromToken` and `amount` are used meaningfully to transfer the `amount` to the SocketGateway and approve the `fromToken` to be spent by the 1inch contract. Everything else is dictated by `swapExtraData`, including even the true amount that is getting swapped. A mishap in the API providing this data payload could cause much less of a token amount to be swapped, a wrong address to receive the swap, and even the wrong destination token to return.

src/swap/oneinch/OneInchImpl.sol:L59-L63

```
// additional data is generated in off-chain using the OneInch API which takes in
// fromTokenAddress, toTokenAddress, amount, fromAddress, slippage, destReceiver, disableEstimate
(bool success, bytes memory result) = ONEINCH_AGGREGATOR.call(
    swapExtraData
);
```

Even the event at the end of the transaction partially refers to the explicitly provided arguments instead of those that actually facilitated the execution of logic

src/swap/oneinch/OneInchImpl.sol:L84-L91

```
emit SocketSwapTokens(
    fromToken,
    toToken,
    returnAmount,
    amount,
    OneInchIdentifier,
    receiverAddress
);
```

As Socket aggregates other solutions, it naturally incurs the trust assumptions and risks associated with its integrations. In some ways, they even stack on top of each other, especially in those Socket functions that batch several routes together – all of them and their associated API calls need to return the correct payloads. So, there is an opportunity to minimize these risks by introducing additional checks into the contracts that would verify the correctness of the payloads that are passed over to the routes and controllers. In fact, creating these payloads within the contracts would allow other systems to integrate Socket more simpler as they could just call the functions with primary logical arguments such as the source token, destination token, and amount.

Recommendation

Consider allocating additional checks within the route implementations that ensure that the explicitly passed arguments match what is being sent for execution to the integrated solutions, like in the above example with the 1inch implementation.

6.5 `NativeOptimismImpl` - Events Will Not Be Emitted in Case of Non-Native Tokens Bridging

Medium

Resolution

Remediated as per the client team in SocketDotTech/socket-ll-contracts#146 by moving the event above the bridging code, making sure events are emitted for all cases, and adding the fix to other functions that had a similar issue.

Description

In the case of the usage of non-native tokens by users, the `SocketBridge` event will not be emitted since the code will return early.

Examples

src/bridges/optimism/l1/NativeOptimism.sol:L110

```
function bridgeAfterSwap(
```

src/bridges/optimism/l1/NativeOptimism.sol:L187

```
function swapAndBridge(
```

src/bridges/optimism/l1/NativeOptimism.sol:L283

```
function bridgeERC20To(
```

Recommendation

Make sure that the `SocketBridge` event is emitted for non-native tokens as well.

6.6 Inconsistent Comments Minor

Resolution

Remediated as per the client team in SocketDotTech/socket-ll-contracts#147.

Description

Some of the contracts in the code have incorrect developer comments annotated for them. This could create confusion for future readers of this code that may be trying to maintain, audit, update, fork, integrate it, and so on.

Examples

src/bridges/stargate/l2/Stargate.sol:L174-L183

```
/**
 * @notice function to bridge tokens after swap. This is used after swap function call
 * @notice This method is payable because the caller is doing token transfer and briding operation
 * @dev for usage, refer to controller implementations
 *      encodedData for bridge should follow the sequence of properties in Stargate-BridgeData struct
 * @param swapId routeId for the swapImpl
 * @param swapData encoded data for swap
 * @param stargateBridgeData encoded data for StargateBridgeData
 */
function swapAndBridge(
```

This is the same comment as `bridgeAfterSwap`, whereas it instead does swapping and bridging together

src/bridges/cbridge/CelerStorageWrapper.sol:L24-L32

```
/**
 * @notice function to store the transferId and message-sender of a bridging activity
 * @notice This method is payable because the caller is doing token transfer and briding operation
 * @dev for usage, refer to controller implementations
 *      encodedData for bridge should follow the sequence of properties in CelerBridgeData struct
 * @param transferId transferId generated during the bridging of ERC20 or native on CelerBridge
 * @param transferIdAddress message sender who is making the bridging on CelerBridge
 */
function setAddressForTransferId(
```

This comment refers to a payable property of this function when it isn't.

src/bridges/cbridge/CelerStorageWrapper.sol:L45-L52

```
/**
 * @notice function to store the transferId and message-sender of a bridging activity
 * @notice This method is payable because the caller is doing token transfer and briding operation
 * @dev for usage, refer to controller implementations
 *      encodedData for bridge should follow the sequence of properties in CelerBridgeData struct
 * @param transferId transferId generated during the bridging of ERC20 or native on CelerBridge
 */
function deleteTransferId(bytes32 transferId) external {
```

This comment is copied from the above function when it does the opposite of storing - it deletes the `transferId`

Recommendation

Adjust comments so they reflect what the functions are actually doing.

6.7 Ether Might Be Sent to Routes by Mistake, and Can Be Stolen Minor

Resolution

The client team has responded with the following note:

> This can happen only if there is an error in API or integration. There are test cases to verify value on API side and we also run an automated testing suite using small amounts after each upgrade to the API before releasing to public. We also work with integrators to test out the flow covering all edge cases before they release. Overall we are fine with taking this risk and relying on rescue function to recover funds while testing.

Description

Most functions of `SocketGateway` are payable, and can receive ether, which is processed in different ways, depending on the routes. A user might send ether to a payable function of `SocketGateway` with a wrong payload, either by mistake or because of an API bug. Let's illustrate the issue with the `performAction` of the 1inch route. However, this can be generalized to other routes.

src/SocketGateway.sol:L90-L97

```
function executeRoute(
    uint32 routeId,
    bytes calldata routeData,
    bytes calldata eventData
) external payable returns (bytes memory) {
    (bool success, bytes memory result) = addressAt(routeId).delegatecall(
        routeData
    );
```

```
function performAction(
    address fromToken,
    address toToken,
    uint256 amount,
    address receiverAddress,
    bytes calldata swapExtraData
) external payable override returns (uint256) {
    uint256 returnAmount;
    if (fromToken != NATIVE_TOKEN_ADDRESS) {
        ...
        {
           ...
            (bool success, bytes memory result) = ONEINCH_AGGREGATOR.call(
                swapExtraData //<-- here we do not use the value
            );
           ...
        }
    } else {
         ....
        (bool success, bytes memory result) = ONEINCH_AGGREGATOR.call{
            value: amount  //<-- here we use the value
        }(swapExtraData);
        ...
    }
  ...
}
```

Assume the user sent some ETH, but sent a payload with `fromToken != NATIVE_TOKEN_ADDRESS` (and the user has already approved the gateway for `fromToken`). Then, the ether is not used in the transaction and remains stuck in the `SocketGateway` contract. This is because the function only executes the part of the code that transfers and swaps ERC-20 tokens, but not the part that handles ether.

Now, suppose another user calls the performAction function with `fromToken == NATIVE_TOKEN_ADDRESS` and provides enough gas to execute the function. Since there is ether stuck in the contract, this user can force the contract to use the stuck ether to execute the swap by sending the exact amount of ether stuck in the contract as the value of the transaction, effectively stealing the funds.

This is why it's important to ensure that ether is only accepted when it is needed and not left stuck in the contract, as it can be vulnerable to theft in future transactions.

One could be tempted to fix the issue by requiring that the gateway balance always equals 0 at the end of the transaction. However, this is not a good idea, as anyone could cause a Denial of Service in the gateway by sending a tiny amount of ETH.

One might also be tempted to fix this issue by requiring that `msg.value == 0` iff `fromToken != NATIVE_TOKEN_ADDRESS`. However, this also poses a problem, as the gateway might execute multiple routes in a "for" loop. This could lead to reverting valid transactions (when both native and non-native tokens are involved).

The best way to solve this issue might be to compare the balance of the gateway before and after the transaction in all relevant functions. The balance should stay the same otherwise, something wrong happened, and we should revert the transaction. This could be implemented by adding a modifier in `SocketGateway`, that compares the balance of the gateway before and after the function call. Below is an example to illustrate the idea.

```
modifier checkGatewayBalance() {
    uint256 initialBalance = address(this).balance;
    _;
    uint256 finalBalance = address(this).balance;
    require(initialBalance == finalBalance, "Gateway balance changed during execution");
}
```

One would also need to introduce a `safeExecuteRoute` function that calls `executeRoute`, but adds the modifier. Note that the other gateway functions calling `executeRoute` in a loop also need to be fixed (such as `swapAndMultiBridge` ...). The `executeRoute` function could be made internal. However, note that one would also need to introduce an admin-protected function that can perform arbitrary delegatecalls on the different routes, without the balance check (such as the current `_executeRoute` function) in case some refunds need to be processed manually (cf. issue 6.1)

6.8 No Event Is Emitted When Invoking a Route Through the `socketGateway` Fallback Function

Minor

Resolution

Remediated as per the client team in SocketDotTech/socket-ll-contracts#152. Further discussion about the scope of events in these cases is still ongoing.

Description

When a route is invoked through `executeRoute`, or `executeRoutes` functions, a `SocketRouteExecuted` event is emitted. However, a route can also be executed by invoking the fallback function of the `socketGateway`. And in that case, no event is emitted. This might impact off-chain systems that rely on those events.

Recommendation

Consider also emitting a `SocketRouteExecuted` event in case the route is invoked through the fallback function

6.9 Unused Error Codes. Minor

Resolution

Remediated as per the client team in SocketDotTech/socket-ll-contracts#148.

Description

SocketErrors.sol has errors that are defined but are not used:

- `error RouteAlreadyExist();`
- `error ContractContainsNoCode();`
- `error ControllerAlreadyExist();`
- `error ControllerAddressIsZero();`

It seems that they were created as errors that may have been expected to occur during the early stages of development, but the resulting architecture doesn't seem to have a place for them currently.

Examples

src/errors/SocketErrors.sol:L12-L19

```
error RouteAlreadyExist();
error SwapFailed();
error UnsupportedInterfaceId();
error ContractContainsNoCode();
error InvalidCelerRefund();
error CelerAlreadyRefunded();
error ControllerAlreadyExist();
error ControllerAddressIsZero();
```

Recommendation

Consider revisiting these errors and identifying whether they need to remain or can be removed.

6.10 Inaccurate Interface. Minor

Resolution

Remediated as per the client team in SocketDotTech/socket-ll-contracts#149.

Description

`ISocketGateway` implies a `bridge(uint32 routeId, bytes memory data)` function, but there is no socket contract with a function like that, including the `SocketGateway` contract.

Examples

src/interfaces/ISocketGateway.sol:L32-L35

```
function bridge(
    uint32 routeId,
    bytes memory data
) external payable returns (bytes memory);
```

Recommendation

Adjust the interface.

6.11 Validate Array Length Matching Before Execution to Avoid Reverts Minor

Resolution

Remediated as per the client team in SocketDotTech/socket-ll-contracts#150 by adding the necessary array length checks.

Description

The Socket system not only aggregates different solutions via its routes and controllers but also allows to batch calls between them into one transaction. For example, a user may call swaps between several DEXs and then perform a bridge transfer.

As a result, the `SocketGateway` contract has many functions that accept multiple arrays that contain the necessary data for execution in their respective routes. However, these arrays need to be of the same length because individual elements in the arrays are intended to be matched at the same indices:

src/SocketGateway.sol:L196-L218

```
function executeRoutes(
    uint32[] calldata routeIds,
    bytes[] calldata dataItems,
    bytes[] calldata eventDataItems
) external payable {
    uint256 routeIdslength = routeIds.length;
    for (uint256 index = 0; index < routeIdslength; ) {
        (bool success, bytes memory result) = addressAt(routeIds[index])
            .delegatecall(dataItems[index]);

        if (!success) {
            assembly {
                revert(add(result, 32), mload(result))
            }
        }

        emit SocketRouteExecuted(routeIds[index], eventDataItems[index]);

        unchecked {
            ++index;
        }
    }
}
```

Note that in the above example function, all 3 different calldata arrays `routeIds`, `dataItems`, and `eventDataItems` were utilizing the same `index` to retrieve the correct element. A common practice in such cases is to confirm that the sizes of the arrays match before continuing with the execution of the rest of the transaction to avoid costly reverts that could happen due to "Index out of bounds" error.

Due to the aggregating and batching nature of the Socket system that may have its users rely on 3rd party offchain APIs to construct these array payloads, such as from APIs of the systems that Socket is integrating, a mishap in just any one of them could cause this issue.

Recommendation

Implement a check on the array lengths so they match.

6.12 Destroyed Routes Eth Balances Will Be Left Locked in `SocketDeployFactory` Minor

Resolution

Remediated as per the client team in SocketDotTech/socket-ll-contracts#151 by adding rescue functions.

Description

`SocketDeployFactory.destroy` calls the `killme` function which in turn self-destructs the route and sends back any eth to the factory contract. However, these funds can not be claimed from the `SocketDeployFactory` contract.

Examples

src/deployFactory/SocketDeployFactory.sol:L170

```
function destroy(uint256 routeId) external onlyDisabler {
```

Recommendation

Make sure that these funds can be claimed.

6.13 Possible Double Spends of `msg.value` in Code Paths That Include More Than One `Delegatecall` Minor

Resolution
The client team has responded with the following note: Adding the recommended CI/CD task to verify that future routes are delegate safe.

Description

The usage of `msg.value` multiple times in the context of a single transaction is dangerous and may lead to loss of funds as previously seen (in a different variation) in the Opyn hack. We were not able to find any concrete instance of the described issue, however, we do see how this pitfall may become an issue in future delegatee contracts.

Examples

Every code path that includes multiple delegatecalls, including:

- `SocketGateway.swapAndMultiBridge`
- the `swapAndBridge` function in all the different route contracts.

Recommendation

Consider implementing this recommendation.

Appendix 1 - Files in Scope

This audit covered the following files:

File	SHA-1 Hash
/src/interfaces/ISocketRequest.sol	d8b481542fa4d3c5ef1cd1a4b49cb4904a05f58e
/src/interfaces/ISocketGateway.sol	d9dfe384c55769f234d34915a53f02c74fa7a8e7
/src/interfaces/ISocketRoute.sol	30b5463db28cbf31a2ed05df722213d80405d6fa
/src/interfaces/ISocketController.sol	ebf9995a6f012633ee70996c8a8041c68b2c504a
/src/interfaces/ISocketBridgeBase.sol	a2c64914a161f5a52a06f4289eaf6152bfa5e53d
/src/controllers/BaseController.sol	37f610b3436923af895eb131921becc0d656fd56
/src/controllers/RefuelSwapAndBridgeController.sol	46686c77fa0ff59c5889c92a51c59207888efda1
/src/controllers/FeesTakerController.sol	02c6a81155bce74cf0d6b09670584792119435ff
/src/errors/SocketErrors.sol	7f530681f134dbcdfd83e18e22b1afe71b7dede1
/src/SocketGateway.sol	f571078fd18a0092736a99271977747512ebb66a
/src/static/RouteIdentifiers.sol	8412534e55dac721ffc7aa438ae3d290bfbed4aa
/src/bridges/BridgeImplBase.sol	4ea3b4268d5c7d9be824a7e9fe73ed51b95c49c1
/src/bridges/cbridge/CelerStorageWrapper.sol	903b8e56d10ce34fa4b5d6a37ca070b19f95912a
/src/bridges/cbridge/CelerImpl.sol	66da779c3f383e136b77fe8877d2e7b5dc24a440
/src/bridges/cbridge/interfaces/ICelerStorageWrapper.sol	cb54f473c4460969a9766b7d8c22aa7df186ece5
/src/bridges/cbridge/interfaces/cbridge.sol	60b4a653355544fcf4f65567e284dc5dc2c509a6
/src/bridges/stargate/l1/Stargate.sol	88a964eae3e0c8d67596280af5a03b76db116684
/src/bridges/stargate/interfaces/stargate.sol	e02de8f547cbcee020a680a7a60bc1eb9f817065
/src/bridges/stargate/l2/Stargate.sol	7a33d727859ba83df9af5f185b5450b51b4e4c14
/src/utils/Ownable.sol	9f7f3a88bf1593e32cdb513e6151d6a5cf3f36e9
/src/swap/SwapImplBase.sol	2555cdac41c3d4ea5d56168365e2b6544ec06288
/src/swap/oneinch/OneInchImpl.sol	28f646473dfe0c31ebee7869f1f69fd88ed07088
/src/deployFactory/DisabledSocketRoute.sol	64a48bc0c2d7afb83d78942147933d12fd7041a3
/src/libraries/LibBytes.sol	ee29b785d7e73b73cd7637a71717223b39fccd03
/src/deployFactory/SocketDeployFactory.sol	5fcd6dd59e683dd6a7429a970d3cf059997f1fe7

File	SHA-1 Hash
/src/libraries/LibUtil.sol	305421b34cc1adf07f16f6d8a3bc03f330357f9a
/src/libraries/Pb.sol	7af0ce1286b28f29e1ab58e60d8c169e74b5dbca

Appendix 2 - Disclosure

ConsenSys Diligence (“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 ConsenSys 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 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 ConsenSys and CD. Such hyperlinks are provided for your reference and convenience only, and are the exclusive responsibility of such web sites’ owners. You agree that ConsenSys and CD are not responsible for the content or operation of such Web sites, and that ConsenSys and 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 ConsenSys and 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. ConsenSys and 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 ConsenSys and CD.