arbitrary./execution

DECENT SECURITY ASSESSMENT

July 5, 2023

Prepared For:

Charlie Durbin and Will Kantaros, Decent

Prepared By:

Arbitrary Execution

Changelog:

June 13, 2023 *Initial report delivered*

July 5, 2023 *Final report delivered*

TABLE OF CONTENTS

EXECUTIVE SUMMARY

This report contains the results of Arbitrary Execution's (AE) assessment of the Decent team's multi-chain transaction platform, The Box. The project aims to create a 1-click checkout solution to execute cross-chain transactions with any token, obviating the need for a user to acquire (either via swap, bridge, or purchase on an exchange) the necessary tokens to complete their desired action.

Two AE engineers conducted this review over a 1-week period, from June 5, 2023 to June 9, 2023. The audited commit hash was `a8db6c6fb2736c20e0aeac8703dd86980e4f0389` in the `decentxyz/UTB` repository. The Solidity files in scope for this audit included all contracts in the `src/` directory. The complete list of files is in Appendix B. These repositories were private at the time of the engagement, so hyperlinks may not work for readers without access.

The team performed a detailed, manual review of the codebase, with a focus on Decent's `Core` contract. This is the second audit of The Box performed by AE. As such, special attention was paid to how issues identified in the first audit were rectified.

The assessment resulted in findings ranging in severity from critical to note (informational). Notes contain observations regarding code hygiene, documentation, and other best practices. The single critical finding allows an attacker to steal other users' ERC20 tokens via a specially-crafted transaction sent to the The Box's `Core` contract.

FIX REVIEW UPDATE

The Decent team has fixed all major issues identified in the engagement. The full breakdown of fixes can be found in the Fixes Summary section.

FIX REVIEW PROCESS

After receiving fixes for the findings shared with the Decent team, the AE team performed a review of each fix. Each pull request was scrutinized to ensure that the associated core issues were addressed, and that no regressions were introduced with the fix. A summary of each fix review can be found in the *Update* section for a finding. For findings that the Decent team chose not to address, the team's rationale is included in the update.

AUDIT OBJECTIVES

AE focuses on common, high-level goals for all security audit engagements. During this engagement, the AE team:

- Identified smart contract vulnerabilities
- Evaluated adherence to Solidity best practices

In addition to the common objectives, there are a wide range of security concerns with a system like The Box, with the security of user funds being paramount. Cross-chain swaps are delicate operations, and once funds leave their chain of origin, a user is placing a tremendous amount of trust in the system. When funds are transferred to The Box for a transaction, a user needs to trust that their funds are secure and the end result of their transaction will be ownership of their desired NFT, as well as a refund of any tokens not used in the purchase. Users also need to be sure that they are getting a fair exchange rate for their tokens. Finally, since The Box will collect fees within its `FeeManager` contract to be withdrawn at a future time, security of these funds must be taken into consideration.

SYSTEM OVERVIEW

The Box facilitates cross-chain asset swaps using Stargate. Stargate is a cross-chain native asset bridge, allowing users to, for example, swap USDC on Ethereum for USDT on BNB in a single transaction on the source chain. The Box's `StargateBridge` will communicate with a `StargateRouter` (not under custody of Decent) to perform these swaps. Any necessary intra-chain swaps are handled using Uniswap pools. Stargate collects a 0.06% fee on cross-chain swaps, and The Box will collect additional fees itself, to be held in its `FeeManager` contract until withdrawn by the contract owner.

SYSTEM COMPONENTS

CORE

The Box `Core` is a smart contract that exposes entry points for the two use cases supported by the The Box protocol (i.e., swapping tokens and bridging tokens across different blockchains). The `Core` contract is effectively a wrapper over the `Dispatcher` and `StargateBridge` contracts, which implement all of the end user functionality.

STARGATEBRIDGE

The `StargateBridge` is a smart contract that implements the cross-chain bridging functionality of The Box protocol. The `StargateBridge` serves two roles. The first role is sending tokens and user payloads from the chain it is deployed on to another chain through a Stargate Router, which is an external protocol responsible for the cross-chain token exchange. The second role is receiving tokens and payloads, sent from other chains, from a Stargate Router. As such, the `StargateBridge` acts as both the The Box protocol's exit point for sending messages to another chain and the entry point for receiving messages from other chains. When the `StargateBridge` receives tokens and a payload from the Stargate Router, it uses the payload to construct a call to a user-supplied address, which provides the

functionality users need to transfer their bridged tokens out of the `StargateBridge` contract to their final destination.

DISPATCHER

The `Dispatcher` is a smart contract that implements the token swapping functionality of The Box protocol. This functionality allows users to purchase NFTs using a token that the seller of the NFT does not accept. To achieve this, the `Dispatcher` uses Uniswap to swap the tokens supplied by the user for the tokens requested by the seller of the NFT. The token swapping functionality also supports executing arbitrary call data after the swap is complete, which allows the swapping functionality to be integrated with other workflows (like purchasing the NFT).

FEEMANAGER

The `FeeManager` is a smart contract responsible for holding fees collected from users as well as calculating those fees for a given transaction. The `FeeManager` contains two state variables, `fee` and `commission`, which are used to calculate the total fees for a given transaction. The `fee` is a flat amount of ether charged for every transaction, and the `commission` is a number of basis points charged as an additional fee on transactions using ether as the payment token. Both `fee` and `commission` can be updated by the contract owner at any time. The contract owner can withdraw the fees held in the `FeeManager` by calling `redeemFees`.

FEEOPERATOR

The `FeeOperator` is a smart contract that serves two main purposes. Firstly, it handles sending fees to the `FeeManager`. Secondly, it is responsible for refunding users any unspent funds. The `FeeOperator` must be assigned a `FeeManager` to fulfill these operations. The main component of the `FeeOperator` is a function modifier, `handleFees`. This function modifier calls into the `FeeManager` to calculate user fees. It then transfers those fees to the `FeeManager` and transfers any leftover funds to the user.

USER CATEGORIES

USERS

Users interact with The Box in two ways, depending on whether they need to bridge assets. Users are expected to call the `swapAndExecute` function when they do not need to bridge assets to another chain in order to purchase an NFT. Users call the `bridgeAndExecute` function when they need to bridge their assets to a different chain in order to purchase an NFT.

PRIVILEGED ROLES

OWNER

There is a fee charged with every NFT purchase and the fees accumulate in the `FeeManager` contract. These fees will periodically be withdrawn via the `redeemFees` function. The contract owner has the privileges to call these functions.

OBSERVATIONS

Overall the code quality and hygiene is excellent, and NatSpec documentation has been added for nearly all components. All major issues identified in the previous audit appear to have been fixed. User fees are no longer collected in the `Core` contract, which was a major source of issues in the previously audited version of The Box.

The critical issue identified concerns the security of user ERC20 tokens once they have been approved to The Box. When a user wishes to use ERC20 tokens for a transaction, they must first approve those tokens to The Box. The Box will then call `transferFrom` on the ERC20 token, transferring the user tokens to the `Core` contract, to then be swapped/bridged as part of an NFT purchase on behalf of the user. While this is a common workflow, there is a vulnerability in The Box that puts those user tokens at risk in the interim between the user approving the funds and The Box calling `transferFrom` to receive them. The root cause of the issue is an arbitrary call in the `Dispatcher` contract. Using a specially-crafted transaction, a malicious user could force The Box to call `transferFrom` on the approved token, transferring those tokens to any address of their choosing.

Arbitrary calls are inherently risky operations and contracts containing them are severely limited in what kinds of things they can do safely. AE recommends avoiding arbitrary calls if at all possible.

VULNERABILITY STATISTICS

Severity	Count
Critical	1
High	0
Medium	0
Low	2
Note	3

FIXES SUMMARY

Finding	Severity	Status
C01	Critical	Fixed in pull request #14
L01	Low	Fixed in pull request #10
L02	Low	Fixed in pull request #11
N01	Note	Fixed in pull request #15
N02	Note	Fixed in pull request #12
N03	Note	Fixed in pull request #13

FINDINGS

CRITICAL SEVERITY

[C01] FUNDS CAN BE STOLEN IN NORMAL OPERATION

In order to use ERC20 tokens with the protocol, users must first approve the tokens to the `Core` contract prior to calling any of the entrypoint functions. Later, when calling `swapAndExecute` or `bridgeAndExecute`, the `_receiveErc20` function uses `safeTransferFrom` to pull the funds into the contract.

Under normal circumstances, this widely used pattern is safe. However, inside the `Core` contract, the `swapAndExecute` function can be used to make a call to an arbitrary address with arbitrary calldata. By using the `target` and `tokenData` parameters, it is possible to reach the arbitrary call in `_swapAndExecute` with user controlled values.

```
(success,) = target.call{value: data.amountOut}(data.payload);
```

Because of this, once an approval is submitted to the `Core`, any other user can send a specially crafted transaction to steal any funds approved from the user's wallet via the `Core` with the same `transferFrom`-style call used in normal operation. As approvals from EOAs cannot be done in a single transaction with a `swapAndExecute` or `bridgeAndExecute` call, it is not possible to avoid the risk of approved funds being stolen.

RECOMMENDATION

Consider removing arbitrary calls from the protocol. If impossible, consider restricting all instances of arbitrary calls from users by limiting who can access them or by requiring specific addresses or function selectors to be used in the calls.

UPDATE

Fixed in pull request #14 (commit hash `8a1a3a8dd86df0ebe26bfa9bb744584eaccaa165`), as recommended.

LOW SEVERITY

[L01] FUNDS CAN ACCUMULATE IN THE `CORE` CONTRACT

When calling the `swap` function on the `StargateRouter`, the fourth parameter is an address to which any extra gas should be refunded. In `Dispatcher.sol`, the `_approveAndBridge` function passes in `payable(this(address))` as the refund address. This will refund unspent gas to the `Core` contract. As a result, funds can accumulate in the `Core` contract that belong to the user.

RECOMMENDATION

Consider refunding excess gas to the user (`msg.sender`).

UPDATE

Fixed in pull request #10 (commit hash `09ec8b8a34ae2159ce1fa0032151d58fc28a7256`), as recommended.

[L02] INCORRECT DOCSTRING

In the `FeeOperator` contract, the NatSpec documentation incorrectly states that it is `An implementation of the ERC1155 multi-token standard.` when it is in fact not. Incorrect documentation can lead to confusion for users and developers.

RECOMMENDATION

Consider removing all cases of incorrect documentation.

UPDATE

Fixed in pull request #11 (commit hash `d0832ff89e3094fba1abbabf8e2f45974e289af7`), as recommended.

NOTE SEVERITY

[N01] HARDCODED VERSION STRING

The `Core` contract inherits the `Version` contract which provides version tracking for the protocol. Rather than setting the version during the constructor, the `Core` contract hardcodes the version into the contract definition.

```
contract Core is Version(1), ICore, FeeOperator, Dispatcher,
IStargateReceiver {
```

Hardcoded version strings are easily overlooked during releases that require version increments.

RECOMMENDATION

Consider passing a version string into the `Core` during deployment.

UPDATE

Fixed. The Decent team removed versioning in pull request #15.

[N02] LACK OF NATSPEC DOCUMENTATION

The `FeeManager` contract lacks proper documentation. Lack of complete documentation makes understanding and interacting with the codebase more difficult.

RECOMMENDATION

The Solidity documentation recommends NatSpec for all public interfaces (everything in the ABI). Consider implementing NatSpec-compliant docstrings for all public and external functions.

A good example is the OpenZeppelin ERC20 contract. It follows the NatSpec guidelines, and provides contract documentation that gives additional information about context and usage.

UPDATE

Fixed in pull request #12 (commit hash `013963869ac77b0e3a74d988168ce637576eef71`), as recommended.

[N03] TODO STATEMENTS IN CODE

There is a `TODO` comment in `Dispatcher.sol`:

- `Dispatcher.sol`, line 58

`TODO` comments are often signs that code is unfinished and may not be ready for deployment. If TODOs are overlooked, there is a risk that deployed code will not match the design specification of the protocol.

RECOMMENDATION

Consider addressing any work remaining in `TODO` statements and removing them from the codebase.

UPDATE

Fixed in pull request #13 (commit hash `42983409a9d627623bf5de8a737ac3bcbbe3b62b`), as recommended.

APPENDIX

APPENDIX A: SEVERITY DEFINITIONS

Severity	Definition
Critical	This issue is straightforward to exploit and is likely to lead to catastrophic impact for client's reputation and can lead to financial loss for client or users.
High	This issue is difficult to exploit and is likely to lead to catastrophic impact for client's reputation and can lead to financial loss for client or users.
Medium	This issue is important to fix and puts a subset of users' data at risk and is possible to lead to moderate financial impact.
Low	This issue is not exploitable in a recurring basis and cannot have a significant impact on execution.
Note	This issue does not pose an immediate risk but is relevant to security best practices.

APPENDIX B: FILES IN SCOPE

```
src
├── base
│   ├── BoxImmutables.sol
│   └── Dispatcher.sol
├── Core.sol
├── interfaces
│   ├── ICore.sol
│   ├── IFeeManager.sol
│   ├── IWrappedToken.sol
│   ├── stargate
│   │   ├── IStargateReceiver.sol
│   │   └── IStargateRouter.sol
│   └── uniswap
│       └── ISwapRouter.sol
├── lib
│   └── CoreStructs.sol
└── utils
    ├── Callbacks.sol
    ├── FeeManager.sol
    ├── FeeOperator.sol
    └── Version.sol
```