Smart Contract Audit

coinspect

OTOCO

OtoCo

Smart Contract Audit

V220520 Prepared for *OtoCo Labs* Ltd. • May 2022

1. Executive Summary

In **May 2022, OtoCo Labs** engaged Coinspect to perform a source code review of **OtoCo Smart Contracts**. The objective of the project was to evaluate the security of the smart contracts.

The following issues were identified during the initial assessment:

High Risk	Medium Risk	Low Risk
1	**1**	**3**
Fixed 0	Fixed 0	Fixed 0

The findings include one high risk issue and one low risk issue related to the maximum amount of possible jurisdictions. Also a medium risk issue was identified that could allow attackers to take over the `OtocoToken` contract by front-running the initialization process.

2. Assessment and Scope

The audit started on **May 2021** and was conducted on the nft-refactoring branch of the git repository at https://github.com/otoco-io/SmartContract as of commit **0833c1b4002c09355fcca40ba06ccd3ce0694592** of **May 13 2022**.
The audited files have the following sha256sum hash:

```
0a60c97a85ce4cf90f7ef3ceb7de1644bc12382f55cd1af5e65043ca37cc78a3  OtoCoPlugin.sol
fb9032e65270f8d298af344f3322bf956694b8df8562ab9794e54a8f73b828c6  utils/IOtoCoMaster.sol
d3ceca6e39aa107299868cfea2fc3c73004793743f0c731fa919dea4a3d974f1
utils/IOtoCoJurisdiction.sol
1bf18baa63be81faa3ae7002b925bf516458470b269acf0710c46886b843e93e  utils/OtoCoToken.sol
f67245e5f56644ad07e468aea76d729087caf22d4a06653016d228859c1438ea  utils/IOtoCoPlugin.sol
79825920dc35aef8995814b253ff150bd63f4d1a505d74dafab9c5c2495a7579  plugins/ENS.sol
e2268c49d371e99b2924fb156a1241912dfffe646011a503fdc3f6b6f0544bb1  plugins/Token.sol
999df07013b33114fde2e6148ec76ccac9b244da3d7f3c88309caa1109036242  plugins/Timestamp.sol
210502743188f318884328b42bbb35b76a2aa57e3755109a5ebd79e9dd860146  plugins/Multisig.sol
fee1888f27a8425c0ee04b99aa2980707f65ee2020d19e75fc778788713a9ca5  plugins/Launchpool.sol
7a530ac6dbe1e09bc0950a30bea6e9d8d029affdbbd673390ef1d5e20803cb5e  OtoCoMaster.sol
9f8d8f4f9d975c608366e245bd30f5662de957b0a9c29c07658eceb17bc5d97b  OtoCoJurisdiction.sol
0456899ec4b2b103ea39decd2d9577240e49574f2456482d9e42c74c1eacb7de
jurisdictions/Delaware.sol
4ea17c38a8af01f0da109210162a6ff2b42057591edf28b93ddbea6dda088fb5
jurisdictions/Unincorporated.sol
ea3fcff85d05f00a3785eb02e0b5f75896fc311478a2d927267af23532db8fb4 jurisdictions/Wyoming.sol
```

Description

OtoCo is an automated company assembly capable of quickly setting up LLCs on the Ethereum blockchain. For this, it utilizes a legal LLC variant called Series LLCs.

A Series LLC depends on a Master LLC. The OtoCo contracts represent an individual Series LLC by minting a NFT for each Series. Because different jurisdictions can have different rules like specific naming conventions, etc, different contracts exist to represent different legal jurisdictions. While there are only three jurisdictions currently implemented in the code, the code allows for up to 65535 jurisdictions. Also, jurisdictions can be added but never removed from the contracts, and this might cause some issues (see OGO-013).

A plug-in mechanism is implemented to add optional features to Series, like ENS services, multisigs and a private token. The plugins are implemented as contracts that can be associated to series in exchange of a fee.

Centralization

The main contract `OtoCoMaster` is upgradeable and owneable.
The `OtoCoToken` implements the token that can be assigned to Series, but this token is not owneable, nor upgradeable.

Roles

The owner of the master contract `OtoCoMaster` is capable of creating Series individually and in batch, adding Jurisdictions, Modifying fees, and withdrawing all fees. The series Owner role is the address that owns the NFT representing the series, it can perform operations like attaching and adding plugins to a series, and also can close the series.

Description of findings

Coinspect found that if the number of jurisdictions surpasses 255, the `OtoCoMaster` contract can no longer mint series in batch mode and always causes a revert of the transaction (OGO-9).

The `OtoCoToken` contract can be taken over by an attacker by front-running the initialization process (OGO-10). The impact of this issue was diminished as medium risk, because the `OtoCoToken` contract is only created via the `Token` plugin contract that always initializes the contract after cloning it.

Regarding low risk issues, it was found that some variables declared in the `OtoCoToken` contract shadow variables in the parent `ERC20` contract. This causes different variables to be used in the parent and inherited contract (OGO-11).

Another low risk issue was found where the contract owner can batch mint series with a timestamp in the past or the future (OGO-12). Also, the function `createBatchSeries()` in the `OtoCoMaster`contract iterates over all the jurisdictions, and if the number of jurisdictions is high the cost of transactions can be rendered prohibitive (OGO-13).

Some informative issues are also reported, concerning documentation and validations, that do not represent security problems (OGO-14, OGO-15)

Finally, we found the included tests to have a reasonable coverage of the contracts code and functionality.

3. Summary of Findings

Id	Title	Total Risk	Fixed
OGO-9	Integer overflow in function createBatchSeries	High	✘
OGO-10	Improper initialization of cloned contract	Medium	✘
OGO-11	Shadowing of state variables	Low	✘
OGO-12	Lack of parameter validation at createBatchSeries	Low	✘
OGO-13	Jurisdiction number causes high gas usage	Low	✘
OGO-14	Inconsistent documentation	Info	✘
OGO-15	Lack of mapping length validations in function createBatchSeries	Info	✘

4. Detailed Findings

OGO-9 Integer overflow in function createBatchSeries

Total Risk	Impact	Location
High	High	`OtoCoMaster.sol:138`
Fixed	Likelihood	
✘	High	

Description

The function `createBatchSeries()` mints a batch of series and adds them to specific state variable counters. When adding the amount of series per jurisdiction, this code is executed:

```
for (uint8 i = 0; i < jurisdictionCount; i++){
        seriesPerJurisdiction[i] = seriesPerJurisdiction[i]+seriesPerJurisdictionTemp[i]
```

However, `jurisdictionCount` is defined as:

```
// Total count of unique jurisdictions
uint16 public jurisdictionCount;
```

As the loop is comparing the variable `i` as `uint8` with `jurisdictionCount` as `uint16`, the transaction reverts if `jurisdictionCount` is over 255, as this will cause an integer overflow in the counter variable `i`.

Additionally, as jurisdictions can only be added and not removed from the contract, if the number of jurisdictions in the contract surpasses 255 the `createBatchSeries()` function will always revert.

Recommendation

Declare the variable `i` as `uint16`.

Status

Not fixed.

OGO-10 Improper initialization of cloned contract

Total Risk	Impact	Location
Medium	Medium	`OtoCoToken.sol:6` `Token.sol:75`
Fixed ✘	Likelihood Medium	

Description

The `Token` contract is a `OtoCoPlugin` that assigns a clone of `OtoCoToken` to a series.

We can see in this declaration:

```
        function initialize (string memory name_, string memory symbol_, uint256 supply_,
address member_) public NotInitialized {
```

Attackers can call the function `initialize` and take over the `OtoCoToken` contract. While this do not affect clones of this contract, the initializer should be disabled in the base contract using the `_disableInitializers()` function in the constructor to automatically lock it when it is deployed:

```
        constructor() {
            _disableInitializers();
        }
```

Recommendation

`OtoCoToken` should implement the initializable pattern, inheriting from OpenZeppelin's `Initializable` contract, and lock the initializer if the contract will only be used through cloning.

Status

Not fixed.

OGO-11	Shadowing of state variables

Total Risk	Impact	Location
Low	Medium	`OtoCoToken.sol:11-17`
Fixed	Likelihood	
✘	Low	

Description

The mappings `_balances, _allowances, and variables _name` and `_symbol` are defined in the base `ERC20` contract and also in the `OtoCoToken` contract. This results in two separate versions of the variables, with functions on the base contract using different variables than functions in the `OtoCoToken` contract, even if they have the same name. This will lead to miscalculations of the token balances and allowances. Fortunately, the `_balances` and `_allowances` mappings are currently not used in the `OtoCoToken` contract so impact is decreased.

Recommendation

Remove any variable names ambiguities between the contracts `ERC20` and `OtoCoToken`.

Status

Not fixed.

OGO-12	Lack of parameter validation at createBatchSeries		
Total Risk **Low**	Impact Medium	Location `OtoCoMaster.sol`	
Fixed ✘	Likelihood Low		

Description

Unlike the `createSeries()` function that sets the series creation timestamp from `block.timestamp`, `createBatchSeries()` do not validate it, and the series timestamp can be set arbitrarily to any time in the past or the future.

Additionally, `createBatchSeries()` also does not validate the series name against restrictions imposed by jurisdictions. According to the documentation:

"For instance, in Wyoming Goldman Sachs LLC could not be used as a standalone name and would have to be referred to as Goldman Sachs LLC, a Series of OtoCo LLC"

But using `createBatchSeries()` the contract owner can add a Series in Wyoming that violates the naming convention.

Recommendation

Validate the creation timestamp and name of every series minted.

Status

Not fixed.

OGO-13 Jurisdiction number causes high gas usage

Total Risk	Impact	Location
Low	Medium	`OtoCoMaster.sol:139`
Fixed	Likelihood	
✘	Low	

Description

This code in `createBatchSeries()` updates the number of series in each jurisdiction:

```
for (uint8 i = 0; i < jurisdictionCount; i++){
    seriesPerJurisdiction[i] = seriesPerJurisdiction[i]+seriesPerJurisdictionTemp[i];
}
```

Because the `jurisdictionCount` variable is declared uint16 and adding jurisdictions is not limited in any way, the contract owner can add up to 65535 jurisdictions. But before this limit is reached, the cost of updating the `seriesPerJurisdiction[]` mapping in the above loop may become prohibitive as it would require over 3 million gas to complete. Also, as you cannot remove unused jurisdictions, this is an ever-increasing cost that the caller must pay every time it invokes the `createBatchSeries()` function, even if it adds only a single series in the batch.

Recommendation

Optimize the loop to only update affected jurisdictions. Implement a lower limit for the number of possible jurisdictions. Alternatively, add a mechanism to remove jurisdictions that are no longer used.

Status

Not fixed.

OGO-14 Inconsistent documentation

Total Risk	Impact	Location
Info	-	`OtoCoToken.sol:21`
Fixed ✘	Likelihood -	

Description

Documentation left over from an older version of the constructor does not match the current code.

Recommendation

Update the documentation with a valid argument description.

Status

Not fixed.

OGO-15	Lack of mapping length validations in function createBatchSeries

Total Risk	Impact	Location
Info	-	`OtoCoMaster.sol:116`
Fixed	**Likelihood**	
✘	-	

Description

The `createBatchSeries()` function receives as arguments four mappings that must be of the same length. It performs length checks on `jurisdiction[]`, `controller[]` and `name[]`, but fails to check the length of the `creation[]` argument. As a consequence the transaction reverts if the `creation[]` mapping length is incorrect without returning an error message.

Recommendation

Add a `require` statement check that the `creation[]` mapping has the same length as the rest of the arguments.

Status

Not fixed.

5. Disclaimer

The information presented in this document is provided "as is" and without warranty. The present security audit does not cover any off-chain systems or frontends that communicate with the contracts, nor the general operational security of the organization that developed the code.