EMERALD SHIELD

Smart Contract Audit Report

Performed for

MFL
Web3 Football Management Game

v1 - October 6th 2022

Disclaimer

The audit reports delivered by Emerald City Labs Inc represents an extensive auditing process intended to help our clients increase the quality of their code while reducing the high level of risk presented by cryptographic tokens and blockchain technology. The Reports do not provide any warranty or representation to any Third-Party in any respect, including regarding the bug-free nature of code, the business model or proprietors of any such business model and the legal compliance of such business model. The Reports should not be used as an endorsement or indictment of the project or team, and does not guarantee the security of the project.

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 or service. The Reports do 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 asset.
Specifically, for the avoidance of doubt, the Reports delivered by Emerald City Labs Inc., should not be represented to any Third-Party as investment advice, nor as an endorsement of the project, team or to the absolute security of the project.

About Us

Emerald City Labs Inc is a private company with its address at 905, 447 Broadway, 2nd Floor, New York, NY, US, 10013

www.ecdao.org

About MFL

Metaverse Football League, or MFL for short, is a web3 football management game that lets users emulates real-world football ecosystems. Players can build and manage their very own football club, compete in leagues and tournaments, and interact with other users to strike deals.

Audit Details

Scope of Audit	6 Contracts & Associated Transactions
Performed By	Jacob Tucker
Git Repo	https://github.com/Metaverse-Football-League/mfl-smart-contracts /tree/feat-clubs/contracts
Git Branch	feat-clubs
Commit Hash	8acd80de0a6736a01841b017de7afd6ded3cb60b

Audit Process

During October 2022, Metaverse Football League (MFL) engaged Emerald City Labs Inc. to conduct an audit of 6 smart contracts that relate to their recently launched dApp. The engagement was technical in nature and the audit was focussed on ensuring the security & efficiency of their Cadence codebase. After an introductory call, MFL provided Emerald City Labs Inc to access to their Github repositories and whitepaper.

This document will be updated to reflect any changes implemented by the team at MFL and when the audit is deemed complete & satisfactory by Emerald Labs Inc. the final version will be published publicly.

Audit Structure

The findings in this document have been structured into the following sections:

- Critical Bugs
- Minor Bugs
- Informational
- Recommendations
- Other Comments & Notes

Critical Bugs

A vulnerability that can disrupt the contract functioning in a number of scenarios, or creates a risk that the contract may be broken.

1.No Critical bugs were found.

Minor Bugs

A vulnerability that affects the desired outcome when using a contract, or provides the opportunity to use a contract in an unintended way.

2. The following **5 Minor bugs** were discovered

MFLPlayer Contract

Item #	CODE REFERENCE	RECOMMENDATION
2.1	Line 243 - NonFungibleToken. Receiver	Should be linked as follows: `self.account.link<&MFLPlayer.Collection{NonFungibleToken.Receiver, NonFungibleToken.CollectionPublic, MetadataViews.ResolverCollection}>(self.CollectionPublicPath, target: self.CollectionStoragePath)`
2.2	Line 178 - PlayerAdminClaim	It seems like you want to use PlayerAdminClaim as a private capability. If this is the case, then you should link the PlayerAdmin to a private path upon contract initialization as follows: `self.account.link<&MFLPlayer.PlayerAdmin{MFLPlayer.PlayerAdminClaim}>(/private/MFLPlayerAdmin, target: self.PlayerAdminStoragePath)`

MFLPackTemplate Contract

Item #	CODE REFERENCE	RECOMMENDATION
2.3	Line 153 - PlayerTemplateAdminClaim	If you want to use PlayerTemplateAdminClaim in a private capability. If so, then PlayerTemplateAdmin should be linked to a private path upon contract initialization as follows: `self.account.link<&MFLPlayer.PlayerTemplateAdmin{MFLPlayer.PlayerTemplateAdminClaim}>(/private/MFLPlaye rTemplateAdmin, target: self.PlayerTemplateAdminStoragePath)`

MFLPack Contract

Item #	CODE REFERENCE	RECOMMENDATION
2.4	Line 208 - NonFungibleToken.Receiver	Should be linked as follows: `self.account.link<&MFLPack.Collection{NonFungibleToken.Receiver, NonFungibleToken.CollectionPublic, MetadataViews.ResolverCollection}>(MFLPack.CollectionPublicPath, target: MFLPack.CollectionStoragePath)`

MFLClub Contract

Item #	CODE REFERENCE	RECOMMENDATION
2.5	Line 620 - NonFungibleToken.Receiver	Should be linked as follows: `self.account.link<&MFLClub.Collection{NonFungibleToken.Receiver, NonFungibleToken.CollectionPublic, MetadataViews.ResolverCollection}>(self.CollectionPublicPath, target: self.CollectionStoragePath)`

Informational

An observation or noted vulnerability that is informational in character but is not effecting any of the code.

3. The following 4 Informational areas were discovered

MFLPlayer Contract

Item #	CODE REFERENCE	RECOMMENDATION
3.1	Line 75	If you want the NFTCatalog to support this contract (so it is displayable in .find and other marketplaces), you will have to support all 5 main views: Display, ExternalURL, NFT Collection Data, NFT Collection Display, and Royalties View.
3.2	Line 225	This is likely intentional, but be aware that if you give a malicious user a PlayerAdmin, they will be able to call the createPlayerAdmin function and spread as many Admins as they want. Potential Solution: Separated into two resources: an Admin and a Minter. The Admin can create as many Minters as they want, and the Minter will be able to create new Players.

MFLPack Contract

Item #	CODE REFERENCE	RECOMMENDATION
3.3	Line 96	Note that if any of the IDs you pass in do not exist in the collection, the whole transaction will fail. If this is intended, then this is fine. Otherwise, you could first check if the id exists in the collection, and then withdraw if so.

MFLPackTemplate Contract

Item #	CODE REFERENCE	RECOMMENDATION
3.4	Line 198	Note that your PackTemplate ids will start at 1. Usually people expect to be indexing from 0, but if this is intentional, that is totally fine.

Recommended Changes

Recommendations to improve efficiency, effectiveness, clarify, maintainability, security, and control based on established best practices.

4. The following 4 recommendations were made

MFLClub Contract

Item #	CODE REFERENCE	RECOMMENDATION
4.1	Line 492	You probably want to add a pre-condition that will panic if a club with the passed in ID already exists. From my understanding, each club id should be completely unique. So you don't want to allow an admin to mint multiple of the same club (and same ID).
4.2	Line 557	You probably want to add a pre-condition that will panic if a squad with the passed in ID already exists. From my understanding, each squad id should be completely unique. So you don't want to allow an admin to mint multiple of the same squad (and same ID).

MFLPackTemplate Contract

Item #	CODE REFERENCE	RECOMMENDATION
4.3	Line 182-183	This line can be improved for the sake of readability / efficiency. Instead of double moving resources, you can simply use the force move operator. The reason this will work is because the IDs will never be overlapping due to it being sequential. Recommended Solution: `MFLPackTemplate.packTemplates[newPackTemplate.id] <-! newPackTemplate`

MFLClub Contract

Item #	CODE REFERENCE	RECOMMENDATION
4.4	Line 245	If it is true that every squad has a unique id, this line can be improved for the sake of readability / efficiency. Instead of double moving resources, you can simply use the force move operator. Recommended Solution: `self.squads[squads[i].id] <-! squads.remove(at: i)`

Other Comments

5. Other comments & questions from the Emerald City team:

All Contracts

Item #	CODE REFERENCE	RECOMMENDATION
5.1	N/A	We noted that all of your emitted events have very little data being broadcasted. In most cases, you just emit an ID. It is not necessary, but I do strongly recommend you add the most amount of data you can to an event, so you can look back on a record of events that have happened in the past, and can access this data if need be.

MFLPackTemplate Contract

Item #	CODE REFERENCE	QUESTION
5.2	Line 71-78	Is there a reason these fields are marked as access(contract)? Do you not want the user to see these fields? Either way, they will always be visible to the user, even with access(contract) scope.