Emerald City Shield

Audit Report

For: Conditional Ownership of NFTs
Performed by: Jacob Tucker

v1
7 October 2022

Table of Contents

Executive Summary

Project Name	Flow Developer Grant - Conditional Ownership of NFTs
Timeline	2 weeks
Method	Project Audit
Scope of Audit	3 Contracts & Associated Transactions
Git Repo link	https://github.com/AlexanderBZ/Flow-Conditional-NFTs
Git Branch	main
Commit hash	b5e40b3481f5df250cba268e52bd1739215d237d

Techniques and Methods

RentNFT

Line 45 - You are trying to return an optional type, but you are also panic'ing on the same line. I would suggest getting rid of the panic and instead returning an optional so the client can handle the nil case instead of aborting inside of a script/transaction.

Line 95 & 97 - Both of these should be changed to access(self). Right now, neither of these pose a security risk, but it is dangerous to allow the contract to be able to access these variables because if you ever exposed a RentList reference to some Admin in the future, they would be able to directly withdraw from people's collections. To increase the confidence in your contract these should be changed.

Line 414 - Is there a need for this function? The Admin could rent an NFT through the normal route (calling the getListedNft() function).

ScrowCollection - It seems like the returnNftToOwner and finishScrow functions rely on an Admin to call them. What are your plans to scale this without needing Admins to run it manually?

TimeBasedExpirationNft

Line 90 - Make sure to implement all required MetadataViews views to be supported in the NFT Catalog. The required views are Display, ExternalURL, NFT Collection Data, NFT Collection Display, and Royalties View.

UseBased

Line 37 - Is it possible for an NFT to already be expired upon creation? You are passing in an expired parameter but I'd imagine all NFTs would not be expired at first. So you should instead set self.expired = false inside the init function.

Line 50 - Make sure to implement all required MetadataViews views to be supported in the NFT Catalog. The required views are Display, ExternalURL, NFT Collection Data, NFT Collection Display, and Royalties View.

Critical Bugs

RentNFT

Payment Type Mismatch - Right now, there is a scary problem with how payment is handled inside a RentList. Anyone can send in a generic @FungibleToken.Vault type, which could be any created Fungible Token on the blockchain. Technically, I

could create my own token worth nothing, and pass that in as payment. With Flow Token this actually should be fine, because when it goes to deposit the payment to the owner it will fail, but we should still flag this if other types of tokens may be involved.
Question: Do you intend to handle all payments only in Flow Token? If so, see Suggested Solution #1. If no, see Suggested Solution #2.

Suggested Solution #1: I would change my capability and vault types to make sure that all types are FlowToken specific. For example, `Capability<&FlowToken.Vault{FungibleToken.Receiver}>` instead of `Capability<&{FungibleToken.Receiver}>`
Suggested Solution #2: Inside the `rentListedNFT` function, add a precondition that checks if the `paymentToRentAndCollateral` and `renterFlowTokenPubCap` both have the same type as `self.flowTokenNftOwnerPubCap`. For example,

```
pre {
      paymentToRentAndCollateral.borrow()!.getType() ==
self.flowTokenNftOwnerPubCap.borrow()!.getType(): "Mismatched payment types."
}
```

Minor Bugs

RentNFT

Line 187 - Inside the `init` function of your RentList resource, I would suggest adding a precondition that uses the `check()` function on both capabilities. You want to make sure the capabilities are valid, so when it's time to borrow them, you don't get unexpected behaviour. The check function is described on this page.
Example:

```
pre {
      _flowTokenNftOwnerPubCap.check(): "This capability is not valid."
      _nftOwnerCap.check(): "This capability is not valid."
}
```

Line 386 - Same advice as above. You should run `check()` on all three capabilities being passed in to the Scrow's `init` function to make sure they are not invalid at time of Scrow creation.

Line 172 & Line 430 - The issue with these functions is that they require a capability to be passed in, of which the NFT must exist in that capability's collection. You could run into a bug where a user decides to invalidate their capability, which will make the function fail every time and prevent you from destroying the listing. Ideally, destroying a listing by an Admin should not be dependent on a user's capability being passed through because the user can always manipulate that.

Line 447 - It's usually best practice to link the full type. Meaning, this should be

```
self.account.link<&RentNFT.ScrowCollection{RentNFT.ScrowPublicCollection}>(/public/
RentNFTScrowCollection, target: /storage/RentNFTScrowCollection)
```

Non-risk Changes

RentNFT

Line 72 - There is a typo inside of your message: “User does not have and NFT inside his Collection”

Line 122 - This is very picky, but saying self.alreadyRented == false can be simplified to !self.alreadyRented

Performance Enhancements

RentNFT

Line 64 - It looks like you are passing in the _nftType but are not using it. Simply delete this parameter.

Line 72 - This check isn’t really needed since it will be handled on the next line anyway. Also, since borrowNFT will never return nil anyway, you can uncomment the code on line 74 and have that be your only precondition. More importantly, this same check is also being performed inside of the init function of your RentList, so it is not entirely needed either. **I would suggest moving line 74 into the RentList init precondition and getting rid of preconditions on the listNFT function.**

Line 77 - Instead of performing a double move operator (and having to destroy on line 87), simply use the force move operator.
Suggested Solution:

```
RentNFT.nftRentList[_nftUuid] <-! create RentList(...all your initializers here…)
```

Questions / Uncertainties / Notes

TimeBasedExpirationNft

Line 26 - In order for an NFT to be expired, this function must be called. What happens if this function is not called, and a client is looking to see if the NFT is expired or not? Technically, the time could be expired, but if nobody called this function it won’t be in the expiredNFTs array.

UseBased

Line 129 - Maybe you don’t want to destroy expired NFTs. Instead, switch their expired field to true, that way you still have a record of the NFT?

How will this contract scale if an Admin has to manually expire NFTs?

Executive Summary

Besides one critical bug, the 3 reviewed contracts are safe to use and will be officially audited once the above notes are fixed.

Disclaimer

Emerald City Schield smart contract audit is not a security warranty, or an investment or legal advice.
The author should not be held accountable for damage arising out of, or in connection with this audit report.
We recommend putting in place a bug bounty program for further analysis of the smart contract by other third parties.