Security Audit

Date : 2023-12-14

Moonvera Solutions

Executive Summary

Type	Smart Contract Audit
Audit Timeline	5 days
Runtime Environment	EVM
Languages	Solidity

Summary of Findings

ID	Name	Description	Severity
M-01	Expiration calculation overflows	expiration calculation reverts when _expirationDays >= 195	Medium
M-02	ERC2981 doesn't handle On-Chain royalties from secondary sales	ERC2981 is Not Equipped to Handle On-Chain Royalties From Secondary Sales	Medium
M-03	Incorrect Access Control added on MvxCollection::mintForOwner()	It's written in the docs that MvxCollection::mintForOwner() can be called by owner but OnlyAdminOrOperator allows operators to call the function as well	Medium
G-01	Use assembly to check for address(0)	Using assembly can help in saving gas while checking for address(0)	Gas
I-01	Incorrect 'Unathorized' spelling	The vulnerability in question is not a security vulnerability per se, but rather a typographical error in the spelling of the custom error `Unathorized`.	Informational
I-02	Magic Numbers Used in Contract	A magical number should be documented and explained	Informational

Findings

[M-01] MvxFactory::updateMember() calculation overflows

Severity: Medium

Location :
https://github.com/moonvera-solution/nft-dex/blob/main/src/MvxFactory.sol#L134

Vulnerability Title: expiration days calculation reverts when _expirationDays >= 195

Description
When calculating the expiration date for a member in the MvxFactory::updateMember() function, the function reverts if the _expirationDays parameter is greater than or equal to 195. Specifically in the expression expiration: **uint40(block.timestamp + (60 * 60 * 24 * (_expirationDays))**). The function reverts because the Solidity compiler (solc) creates a temporary variable to store a value with the least possible uint type, which, in the case of 60 * 60 * 24, is uint24. Since values are typecasted to the maximum variable type during multiplication, values above **(2^24 - 1) / 86400 = 194** will revert the function.

Impact
Even though its stated in the documentation that the expiration date is capped to 10 days , if in the future the protocol decides to increase the expiration date for example adding a tier system in the protocol so that higher tier members enjoy a higher expiration date , the protocol will be forced to make a system with the max days allowed as 194.

Recommendation
Consider modifying the expression to: **uint40(block.timestamp + (60 * 60 * 24 * uint40(_expirationDays)))**. This typecasts the variable so that its large enough to hold the values for the multiplication

POC
In the contract **MvxFactoryMemberDiscountTest** add this function :

```
    function test_member_update() public {
        Vm.Wallet memory member = vm.createWallet("member");
        uint16 _memberDiscount = 2000; // 20% discount
        uint72 _deployFee = 0.5 ether;
        uint256 _feeAfterDiscount = _deployFee -
_deployFee.mulDiv(_memberDiscount, 10_000); // basis points
        factory.updateMember(member.addr, address(0x0), _deployFee, 0,
_memberDiscount, 195);
```

```
        vm.startPrank(member.addr);
        vm.deal(member.addr, 1 ether);
        address _clone = _factoryCreate(factory, member.addr,
_feeAfterDiscount);
        vm.stopPrank();

        assert(_clone != address(0x0));
        assert(address(factory).balance < _deployFee);
        assertEq(address(factory).balance, _feeAfterDiscount);
    }
```

```
Running 3 tests for test/foundry/invariant/MvxFactoryMemberDiscountTest.sol:MvxFactoryMemberDiscountTest
[PASS] test_fuzz_member_discount((address,uint256,uint256,uint256),uint16,uint72) (runs: 256, μ: 1848834, ~: 1848834)
[PASS] test_member_discount() (gas: 1846057)
[FAIL. Reason: panic: arithmetic underflow or overflow (0x11)] test_member_update() (gas: 12277)
Test result: FAILED. 2 passed; 1 failed; 0 skipped; finished in 1.30s
```

Similar Findings :
https://solodit.xyz/issues/m-05-expiration-calculation-overflows-if-call-option-duration-195-days-code4rena-cally-cally-contest-git

Other instances
:https://github.com/moonvera-solution/nft-dex/blob/main/src/MvxCollection.sol#L73

[M-02] ERC2981 is Not Equipped to Handle On-Chain Royalties From Secondary Sales

Severity: Medium

Vulnerability Title: Documentation mentions that ERC2981 is used to gain royalties from secondary sales but it doesn't handle getting royalties from secondary sales

Description
A bug has been identified in the contract/documentation which states that royalty should be equipped to handle revenue generated on a collection's secondary sales. Currently, MvxCollection allows the collection owner to receive a fee on each token mint, but there is no existing implementation which allows the owner of a collection to receive fees on secondary sales.
Impact

The absence of an implementation to allow the collection owner to receive fees on secondary sales can lead to a loss of potential revenue for the owner and may not align with the intended functionality outlined in the documentation.

Recommendation
To address this issue, it is recommended to update the MvxCollection.sol contract to include a robust implementation that enables the collection owner to receive royalties on secondary sales. This implementation should be thoroughly tested to ensure that it functions as intended and complies with the specifications outlined in the contract/documentation.

[M-03] Incorrect Access Control added on MvxCollection::mintForOwner()

Severity: Medium

Location :
https://github.com/moonvera-solution/nft-dex/blob/main/src/MvxCollection.sol#L85

Vulnerability Title
Its written in the docs that MvxCollection::mintForOwner() can be called by owner but OnlyAdminOrOperator Modifier allows operators to call the function as well

Description
The documentation states that MvxCollection::mintForOwner() can be called by the owner, but the OnlyAdminOrOperator modifier allows operators to call the function as well. This situation breaks the governance of the protocol, as it is not permitted for operators to call that function. To ensure correct access control and adhere to the specified governance, it is recommended to consider removing the OnlyAdminOrOperator modifier and instead adding the OnlyAdmin modifier in its place.

Impact
The current implementation, where the OnlyAdminOrOperator modifier allows operators to call the MvxCollection::mintForOwner() function, can lead to a governance breach and potential misuse of privileges. This deviation from the intended access control mechanism may compromise the protocol's security and operational integrity.

Recommendation
To address this issue, it is advisable to remove the OnlyAdminOrOperator modifier from the MvxCollection::mintForOwner() function and replace it with the OnlyAdmin modifier. This adjustment will ensure that only the owner and authorized administrators can call the function, aligning with the specified governance and access control requirements.
By making this change, the protocol will maintain the integrity of its access control mechanisms and prevent unauthorized access to critical functions, thereby upholding the intended governance structure and security measures

[G-01] Use Assembly to check for address(0)

Severity: Gas

Vulnerability Title: Gas can be saved by using Assembly

Location
https://github.com/moonvera-solution/nft-dex/blob/main/src/MvxFactory.sol#L104
https://github.com/moonvera-solution/nft-dex/blob/main/src/MvxFactory.sol#L126
https://github.com/moonvera-solution/nft-dex/blob/main/src/MvxFactory.sol#L205
https://github.com/moonvera-solution/nft-dex/blob/main/src/MvxCollection.sol#L166
https://github.com/moonvera-solution/nft-dex/blob/main/src/abstracts/MintingStages.sol#L101

Description
One way to optimize gas usage is by using assembly for checking for null address checks. This bypasses some checks saving some gas

Impact
By using assembly for null address checks, you can:
Save gas costs associated with unnecessary operations and transfers.
Improve the overall efficiency of your smart contract code.

Reduce the gas footprint of your transactions, which can lead to cost savings for users.

Recommendation
Identify the parts of your code where null address checks are performed.
Replace the null address checks with assembly code that directly checks for the null address. Use the assembly keyword to write the assembly code, which allows you to bypass certain Solidity restrictions and optimize gas usage

[I-01] Incorrect 'Unathorized' spelling

Severity: Informational

Vulnerability Title: Incorrect 'Unathorized' Spelling

Location
https://github.com/moonvera-solution/nft-dex/blob/main/src/MvxFactory.sol#L49

Description
The vulnerability in question is not a security vulnerability per se, but rather a typographical error in the spelling of the custom error `Unathorized`. The correct spelling should be `Unauthorized`. While this does not pose a direct risk to the contract's security or functionality, it can lead to confusion and a lack of professionalism in the codebase. Error messages are a critical part of the developer interface for any smart contract, and misspellings can cause misunderstandings when they are triggered and read by developers or users interacting with the contract.

Impact
The impact of this typographical error is primarily on the clarity and quality of the code. It does not affect the execution or security of the contract. However, it is important to maintain a high standard of code quality, and such errors should be corrected to ensure clear communication and to avoid any potential confusion in the future.

Recommendation
The recommended action is to correct the spelling of the custom error from

`Unathorized` to `Unauthorized`. This change should be made wherever the error is defined and used within the contract to maintain consistency. After making the correction, thorough testing should be conducted to ensure that the error messages work as expected and that no references to the misspelled error remain in the codebase.

[I-02] Magic Numbers Used in Contract

Severity: Informational

Location:
https://github.com/moonvera-solution/nft-dex/blob/main/src/abstracts/MintingStages.sol#L130C43-L130C43
https://github.com/moonvera-solution/nft-dex/blob/main/src/abstracts/MintingStages.sol#L130C43-L130C43
https://github.com/moonvera-solution/nft-dex/blob/main/src/abstracts/MintingStages.sol#L130C43-L130C43
https://github.com/moonvera-solution/nft-dex/blob/main/src/abstracts/MintingStages.sol#L130C43-L130C43
https://github.com/moonvera-solution/nft-dex/blob/main/src/MvxCollection.sol#L222

Description
Using magic numbers in contracts can lead to maintainability and readability issues. To avoid these problems, it is recommended to use constants that are documented and well-defined in the documentation and the contract. This approach ensures that your code remains clear and easy to understand for both developers and users.
Impact
By using documented and well-defined constants instead of magic numbers in your contract:
You improve the maintainability and readability of your code.
You ensure that your code adheres to best practices and standards.
You make it easier for developers to understand and debug your contract.
Recommendation

To replace magic numbers with documented constants in your contract, follow these steps:
Identify the parts of your code where you are using magic numbers.
Replace the magic numbers with documented constants.
Update your documentation to include the new constants and their descriptions.
