Protocol Audit Report

Version 1.0

Ekene Audits

March 12, 2025

Protocol Audit Report

Ekene

March 12, 2025

Prepared by: Ekene Audits Lead Auditors: - Ekene

Table of Contents

Protocol Summary

InheritableSmartContractWallet 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.

Disclaimer

The Ekene Audit team makes all effort to find as many vulnerabilities in the code in the given time period, but holds no responsibilities for the findings provided in this document. A security audit by the team is not an endorsement of the underlying business or product. The audit was time-boxed and the review of the code was solely on the security aspects of the Solidity implementation of the contracts.

Risk Classification

		Impact		
		High	Medium	Low
Likelihood	High	H	H/M	M
	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

Scope

Roles

Executive Summary

Issues found

Findings

High

Medium

Low

Informational

Gas

Audit Findings Summary

This document details the audit findings for the contract, organized by severity. Each finding includes a title, summary, vulnerability details, impact assessment, tools used, and recommendations. Collapsible code sections are provided where code samples are included.

Table of Contents

High Severity Findings

1. Potential Denial of Service (DOS) Due to Blacklisted Address

Summary:
If any beneficiary's address is blacklisted in the payment asset, functions such as `buyOutEstateNFT` and `withdrawInheritedFunds` will revert inside a loop—effectively locking out these functionalities.

Vulnerability Details:
Within the functions, a revert is triggered inside a loop when encountering a blacklisted address. This can cause the entire function to become uncallable.

Test & Code Example

```
1   function test__One_Address_Can_lockup_buyOutEstateNFT() public {
2       _addDelegates();
3       im.createEstateNFT(_description, asset_value, address(usdc));
4       vm.startPrank(user3);
5       usdc.mint(user3, 1000e18);
6       usdc.approve(address(im), 1000e18);
7       vm.warp(block.timestamp + 95 days);
8       im.inherit();
9       vm.expectRevert();
10      im.buyOutEstateNFT(1);
11  }
```

Impact:
High – The function can be intentionally locked, causing a denial of service.

Tools Used:
slither, aderyn, foundry

Recommendations:
When iterating over beneficiaries, use a try-catch block to handle transfer failures rather than reverting the entire loop.

Diff Recommendation

```
    for (uint256 i = 0; i < beneficiaries.length; i++) {
        if (msg.sender == beneficiaries[i]) {
            return;
        } else {
-           IERC20(assetToPay).safeTransfer(beneficiaries[i], finalAmount / divisor);
+           try IERC20(assetToPay).safeTransfer(beneficiaries[i], finalAmount / divisor) {
+           } catch {
+               emit TransferFailed(beneficiaries[i], finalAmount / divisor);
+           }
        }
    }
```

---

2. Inaccurate Beneficiary Index Retrieval in `_getBeneficiaryIndex`

Summary:
The `_getBeneficiaryIndex` function returns zero when a beneficiary address is not found, which is problematic because index 0 is a valid position. This misbehavior may lead to unintended deletions.

Vulnerability Details:
When the beneficiaries array is not empty, a non-existent address will return index 0. This value is then used in `removeBeneficiary`, potentially deleting the wrong beneficiary.

Test Example

```
function test__getBeneficiaryIndex__will_return_zero_if_address_is_not_a_delegate() public inheritanceSetUp {
    assertEq(im._getBeneficiaryIndex(address(60)), 0);
}
```

Impact:
High – Incorrect beneficiary removal can lead to loss of funds or mismanagement of estate distributions.

Tools Used:
slither, aderyn, foundry

Recommendations:
Introduce a custom exception (e.g., `InheritanceManager__Not_A_Beneficiary()`) if the address is not found in the beneficiaries list.

---

3. Insecure Beneficiary Removal Leaving Zero Address

Summary:
The `removeBeneficiary` function does not fully delete a beneficiary; it only replaces the entry with a zero address, which can still be interpreted as a valid beneficiary.

Vulnerability Details:
Using the `delete` keyword only resets the value to the zero address rather than removing the element from the array.

Test & Code Example

```
1   function
       test__removeBeneficiary_only_replaces_address_with_zero_address()
       public inheritanceSetUp {
2       vm.startPrank(address(this));
3       uint user3Index = im._getBeneficiaryIndex(user3);
4       im.removeBeneficiary(user3);
5       console.log('here is the index ', user3Index);
6       assertEq(user3Index, im._getBeneficiaryIndex(address(0)));
7       vm.stopPrank();
8   }
```

Impact:
High – Leaving a zero address in the array could affect inheritance distributions and contract logic.

Tools Used:
slither, aderyn, foundry

Recommendations:
Replace the beneficiary to be removed with the last element in the array and then call `.pop()` to delete the last entry.

Diff Recommendation

```
  function removeBeneficiary(address _beneficiary) external onlyOwner {
      uint256 indexToRemove = _getBeneficiaryIndex(_beneficiary);
-     delete beneficiaries[indexToRemove];
+     beneficiaries[indexToRemove] = beneficiaries[beneficiaries.length
    - 1];
+     beneficiaries.pop();
  }
```

---

4. Missing Deadline Update in Critical Functions

Summary:
The deadline is not updated in functions such as `createEstateNFT` and `contractInteractions`, potentially leading to premature asset inheritance.

Vulnerability Details:
Failure to update the deadline after these transactions can make an account appear dormant even when it is active, causing unintended behaviors.

Code Example

```
  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;
      }
  }
```

Impact:
High – Incorrect deadline handling may trigger premature inheritance events.

Tools Used:
slither, aderyn, foundry

Recommendations:
Incorporate a deadline update (e.g., by calling `_setDeadline()`) at the end of these functions.

Diff Recommendation

```
  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();
  }
```

---

5. Lack of Proper ETH Handling in Wallet Contract

Summary:
The contract is intended to function as a wallet; however, it lacks the mechanisms to handle ETH, which may prevent the reception of funds needed for transaction fees.

Vulnerability Details:
Without implemented `receive` and `fallback` functions, any ETH sent to the contract will be rejected.

Test Example

```
  function test__contract_can_not_handle_eth() public {
      vm.deal(user1, STARTING_BALANCE);
      vm.prank(user1);

      (bool success,) = address(im).call{value: 10e18}("");
      assert(success == false);
  }
```

Impact:
High – Inability to handle ETH could compromise the operational capabilities of the wallet.

Tools Used:
slither, aderyn, foundry

Recommendations:
Implement both `receive()` and `fallback()` functions to enable proper ETH handling.

Code Recommendation

```
receive() external payable {
    // handle incoming ETH
}

fallback() external payable {
    // handle fallback calls
}
```

Medium Severity Findings

6. Unrestricted Trustee Appointment Enabling Asset Price Manipulation

Summary:
Any beneficiary can appoint a trustee, which may allow them to assign a favorable trustee and manipulate the asset price for a potential buyout.

Vulnerability Details:
The `appointTrustee` function does not restrict who can be appointed as trustee. This allows a beneficiary to potentially reduce the asset's price to an almost negligible amount and then execute a buyout.

Test Example

```
function test__any_delegate_can_assign_trustee_to_change_asset_value
    () public inheritanceSetUp {
    vm.startPrank(user1);
    im.appointTrustee(user2);
    vm.stopPrank();
    vm.prank(user2);
    im.setNftValue(1, 1e16);

    assertEq(im.getNftValue(1), 1e16);
}
```

Impact:
Medium – This flaw can lead to asset mispricing and potential financial abuse.

Tools Used:
slither, aderyn, foundry

Recommendations:
Implement a consensus mechanism ensuring that a majority of beneficiaries approve any trustee assignment.

---

7. Unintended Truncation in `buyOutEstateNFT` Function

Summary:
The loop in `buyOutEstateNFT` incorrectly exits (truncates execution) when the caller is found before reaching the last index, preventing the full execution of intended logic.

Vulnerability Details:
The early return within the loop causes incomplete processing if the caller is not the last element in the beneficiaries array.

Test & Code Example

```
    if (msg.sender == beneficiaries[i]) {
        return;
    }
```

Impact:
Medium – Leads to incomplete execution, affecting asset buyout logic.

Tools Used:
slither, aderyn, foundry

Recommendations:
Replace the **`return`** statement with **`continue`** so that the loop can iterate over all beneficiaries.

Diff Recommendation

```
     if (msg.sender == beneficiaries[i]) {
-        return;
+        continue;
     }
```

---

8. Missing Checks for Duplicate and Zero Addresses in Beneficiary Management

Summary:
There are no validations to prevent duplicate beneficiaries or the addition of zero addresses in functions

like `inherit` and `addBeneficiery`.

Vulnerability Details:
Allowing duplicate entries or a zero address can lead to unintended consequences during inheritance distributions.

Impact:
Medium – Can lead to logical errors and mismanagement of beneficiary data.

Tools Used:
slither, aderyn, foundry

Recommendations:
Introduce proper validations to reject duplicate addresses and zero addresses during beneficiary additions.

Low Severity Findings

9. No Event Emission on State Change in `appointTrustee`

Summary:
The `appointTrustee` function updates the state without emitting an event, reducing effective offchain communication.

Vulnerability Details:
The absence of an event may hinder off-chain tracking and notification processes.

Code Example

```
1    function appointTrustee(address _trustee) external
        onlyBeneficiaryWithIsInherited {
2        trustee = _trustee;
3    }
```

Impact:
Low – Although not critical, it impairs the audit trail.

Tools Used:
slither, aderyn, foundry

Recommendations:
Emit an event after updating the trustee to improve transparency.

Diff Recommendation

```
1    function appointTrustee(address _trustee) external
        onlyBeneficiaryWithIsInherited {
2        trustee = _trustee;
3    +    emit InheritanceManager__AppointedTrustee();
4    }
```

---

10. Precision Loss in Calculation within `buyOutEstateNFT`

Summary:
The calculation in `buyOutEstateNFT` may suffer from precision loss, particularly when dealing with ERC20 tokens with small decimal values.

Vulnerability Details:
Using integer division in the expression
`uint256 finalAmount = (value / divisor)* multiplier;`
can lead to rounding errors because Solidity rounds down on division.

Impact:
Low – Results in minor discrepancies that could be critical when dealing with precise financial calculations.

Tools Used:
slither, aderyn, foundry

Recommendations:
Either restrict the asset types to those with appropriate decimal precision or adjust the calculation method to maintain precision.

---

11. Inefficient `onlyBeneficiaryWithIsInherited` Modifier

Summary:
The modifier `onlyBeneficiaryWithIsInherited` uses an unbounded loop with an out-of-bound exception risk, leading to high gas usage and potential vulnerabilities.

Vulnerability Details:
Iterating with a loop that exceeds the beneficiaries array bounds is inefficient and may lead to unintended errors.

Code Example

```
modifier onlyBeneficiaryWithIsInherited() {
    uint256 i = 0;
    while (i < beneficiaries.length + 1) {
        if (msg.sender == beneficiaries[i] && isInherited) {
            break;
        }
        i++;
    }
    _;
}
```

Impact:
Low – While not immediately dangerous, it is inefficient and could be optimized.

Tools Used:
slither, aderyn, foundry

Recommendations:
Replace the current logic with custom errors and more efficient checks to avoid out-of-bound access.

This concludes the organized and enhanced audit findings report. Each recommendation should be carefully considered and implemented where applicable to improve the security and robustness of the contract.