Mafia Take Down Audit Report

Prepared by: Heeze Lead Auditor:

- Heeze

Table of Contents

- Mafia Take Down Audit Report
- Table of Contents
- Risk Classification
- Audit Scope Details
- Protocol Summary
  - Roles
  - Issues Found
  - High
    - [H-1] The function `MoneyShelf::depositUSDC` uses arbitrary `from` and `to` address.
    - [H-2] Calls to the `MoneyVault::withdrawUSDC` function will always fail as the caller needs to have permissions to call it.
    - [H-3] The `Shelf` contract which is inherited by both `MoneyVault` and `MoneyShelf` contract keeps track of every deposit using the `bank` mapping.
    - [H-4] The funds in the `MoneyShelf` contract are not moved to `MoneyVault` during migration.
  - Medium
    - [M-1] The `Kernel` upgrades the `MoneyShelf` to `MoneyVault` but doesn't disable the `MoneyShelf` contract.
  - Low
    - [L-1] Unchecked transfers
    - [L-2] Missing checks for `address(0)` when assigning values to address state variables
    - [L-3] `public` functions not used internally could be marked `external`
  - [L-4]: PUSH0 is not supported by all chains
  - [L-5]: Modifiers invoked only once can be shoe-horned into the function

Risk Classification

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

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

Audit Scope Details

- Commit Hash: XXX
- Files in scope:

```
├── script
│   ├── Deployer.s.sol
│   ├── EmergencyMigration.s.sol
├── src
│   ├── CrimeMoney.sol
│   ├── modules
│   │   ├── MoneyShelf.sol
│   │   ├── MoneyVault.sol
│   │   ├── Shelf.sol
│   │   └── WeaponShelf.sol
│   ├── policies
│   │   └── Laundrette.sol
```

Protocol Summary

Policies: These are external-facing contracts that handle access control and input validation.

Laundrette: Anyone can deposit USDC to receive CrimeMoney, but only gang members and the godfather can withdraw USDC. Users must approve the MoneyShelf contract before making a deposit.

The godfather is the only one who can control incoming weapons and assign them to gang members. The contract 'Kernel.sol' administers roles and allows the godfather to retrieve the admin role when needed.

WeaponShelf: This contract keeps count of the available weapons per member.

MoneyShelf: This contract is in charge of keeping USDC and minting/burning the CrimeMoney. Users need to approve this contract before depositing.

MoneyVault: In case of any issue, MoneyShelf is updated to protect the money. Only the GodFather can withdraw from this contract.

CrimeMoney: A stablecoin pegged to USDC deposited in MoneyShelf.

Roles

- GodFather: Owner, has all the rights.
- GangMember:
  - Deposit USDC and withdraw USDC in exchange for CrimeMoney
  - Transfer CrimeMoney between members and godfather.
  - Take weapons that GodFather assigned to the member.
- External users: can only call view functions and deposit USDC.

Issues Found

Severity	Number of issues found
High	4
Medium	1
Low	5
Total	10

High

[H-1] The function `MoneyShelf::depositUSDC` uses arbitrary `from` and `to` address.

Description: Passing an arbitrary `from` address to `transferFrom` (or `safeTransferFrom`) can lead to loss of funds, because anyone can transfer tokens from the `from` address if an approval is made.

Impact: If an approval has been made, an attacker can call the `MoneyShelf::depositUSDC` function and pass the user's address as the `account` parameter and their address as the `to` parameter, hence sending user's funds to the `MoneyShelf` and stealing the tokens the user is meant to recieve for depositing USDC (CrimeMoney tokens).

Proof of Concept:

- User approves `MoneyShelf` to spend thier USDC
- Attacker calls the `MoneyShelf::depositUSDC` function and pass the user's address as the `account` parameter and their address as the `to` parameter.
- Attacker steals users crimeeMoney tokens.

▶ Code

```
    function test_canHijackUserFunds() public {
          vm.startPrank(godFather);
          usdc.transfer(user, 100e6);
          vm.stopPrank();

          vm.startPrank(user);
          usdc.approve(address(moneyShelf), 100e6);
          vm.stopPrank();

          vm.startPrank(attacker);
          laundrette.depositTheCrimeMoneyInATM(user, attacker, 100e6);
          assertEq(usdc.balanceOf(user), 0);
          assertEq(usdc.balanceOf(address(moneyShelf)), 100e6);
          assertEq(crimeMoney.balanceOf(attacker), 100e6);
          assertEq(crimeMoney.balanceOf(user), 0);
      }
```

Recommended Mitigation: Use `msg.sender` in place of the `account` and `to` parameters.

```diff
- function depositUSDC(address account, address to, uint256 amount)
external {
-       deposit(to, amount);
-       usdc.transferFrom(account, address(this), amount);
-       crimeMoney.mint(to, amount);
+ function depositUSDC(uint256 amount) external {
+       deposit(msg.sender, amount);
+       usdc.transferFrom(msg.sender, address(this), amount);
+       crimeMoney.mint(msg.sender, amount);
    }
    }
```

[H-2] Calls to the `MoneyVault::withdrawUSDC` function will always fail as the caller needs to have permissions to call it.

Description: The `MoneyVault::withdrawUSDC` function will always revert when called by the `godFather` because it can only be called by a smart cntract with the appropriate permissions.

Impact: When the `MoneyShelf` module is upgraded to the `MoneyVault` module during an emergency, the `godFather` will be unable to withdraw the USDC in the contract as the call to `MoneyVault::withdrawUSDC` will always revert leading to the USDC being stuck.

Proof of Concept:

▶ Code

```
    function test_withdrawFromMoneyVault() public {
        joinGang(address(0));
        uint256 godFatherStartingBal = usdc.balanceOf(godFather);
        vm.startPrank(godFather);
        usdc.approve(address(moneyShelf), 500e6);
        laundrette.depositTheCrimeMoneyInATM(godFather, godFather, 250e6);
        laundrette.depositTheCrimeMoneyInATM(godFather, godFather, 250e6);
        vm.stopPrank();

        EmergencyMigration migration = new EmergencyMigration();
        MoneyVault moneyVault = migration.migrate(kernel, usdc,
crimeMoney);


assertEq(address(kernel.getModuleForKeycode(Keycode.wrap("MONEY"))),
address(moneyVault));

        vm.startPrank(godFather);
        //-------------Reverts-------------
        moneyVault.withdrawUSDC(godFather, godFather, 500e6);
        vm.stopPrank();

        assertEq(usdc.balanceOf(godFather), godFatherStartingBal);
    }
```

Recommended Mitigation: Either the `godFather` is given the required permissions to call the `MoneyVault::withdrawUSDC` function, or some other implementation that allows the `godFather` to be able to call the `MoneyVault::withdrawUSDC` function.

[H-3] The `Shelf` contract which is inherited by both `MoneyVault` and `MoneyShelf` contract keeps track of every deposit using the `bank` mapping.

Description: The `Shelf` contract is an abstract contract inherited by both `MoneyVault` and `MoneyShelf` and it keeps track of every deposit made using the `bank` mapping and uses it in the withdraw function hence a `gangMember` or the `godFather` can only withdraw as much as they deposit.

Impact: The contract allows anyone to deposit but only allows the `godFather` and `gangMembers` to withdraw but due to the `bank` mapping that only allows withdrawal on as much as has been deposited, the `gangMembers` and the `godFather` will only be able to withdraw as much as they deposited which means anyone who isn't the `godFather` or `gangMembers` and deposits, their funds cannot be withdrawn leading to it being stuck in the contract.

Proof of Concept:

▶ Code

```
function test_godfatherCantWithdrawUserTokens() public {
        vm.startPrank(godFather);
        usdc.transfer(user, 100e6);
        vm.stopPrank();
        uint godFatherStartingBalance = usdc.balanceOf(godFather);
        vm.startPrank(user);
        usdc.approve(address(moneyShelf), 100e6);
        laundrette.depositTheCrimeMoneyInATM(user, user, 100e6);
        vm.stopPrank();
        assertEq(usdc.balanceOf(user), 0);
        assertEq(usdc.balanceOf(address(moneyShelf)), 100e6);
        assertEq(crimeMoney.balanceOf(user), 100e6);

        joinGang(address(0));
        vm.startPrank(godFather);
        // --------------Reverts----------------------
        laundrette.withdrawMoney(godFather, godFather, 100e6);
        assertEq(usdc.balanceOf(godFather), godFatherStartingBalance);
        assertEq(usdc.balanceOf(address(moneyShelf)), 0);
        assertEq(crimeMoney.balanceOf(godFather), 0);
    }
```

Recommended Mitigation: One way of addressing this could be by using different deposit functions for users (which doesn't track their deposits), gangMembers and the godFather such that the godFather can withdraw any amount of tokens avaliable at any given time.

[H-4] The funds in the `MoneyShelf` contract are not moved to `MoneyVault` during migration.

Description: During an emergency the `MoneyShelf` contract is upgraded to the `MoneyVault` to protect the funds from the justice system or any other gang. But the funds are not moved to `MoneyVault` during or after upgrade.

Impact: After an upgrade is done the `MoneyVault` contract will still be empty and the `godFather` will not be able to withdraw the USDC tokens as there will be no tokens in the vault.

Proof of Concept:

▶ Code

```
 function test_moveFundsWhenMigrating() public {

assertEq(address(kernel.getModuleForKeycode(Keycode.wrap("MONEY"))),
address(moneyShelf));

        vm.startPrank(godFather);
        usdc.transfer(user, 100e6);
        vm.stopPrank();

        vm.startPrank(user);
        usdc.approve(address(moneyShelf), 100e6);
        laundrette.depositTheCrimeMoneyInATM(user, user, 100e6);
        assertEq(usdc.balanceOf(user), 0);
        assertEq(usdc.balanceOf(address(moneyShelf)), 100e6);
        assertEq(crimeMoney.balanceOf(user), 100e6);
        vm.stopPrank();

        EmergencyMigration migration = new EmergencyMigration();
        MoneyVault moneyVault = migration.migrate(kernel, usdc,
crimeMoney);


assertEq(address(kernel.getModuleForKeycode(Keycode.wrap("MONEY"))),
address(moneyVault));
        assertEq(usdc.balanceOf(address(moneyVault)), 100e6);
    }
```

Recommended Mitigation: During the migration the `kernel` contract should move the funds from `MoneyShelf` to `MoneyVault` contract.

Medium

[M-1] The `Kernel` upgrades the `MoneyShelf` to `MoneyVault` but doesn't disable the `MoneyShelf` contract.

Description: When the `Kernel` contract upgrades/migrates the `MoneyShelf` module to `MoneyVault` it doesn't disable the functions and functionality of the `MoneyShelf` module.

Impact: After the migration anyone can still deposit into the `MoneyShelf` contract and `gangmembers` can still still withdraw their funds from the `MoneyShelf` contract.

Proof of Concept:

▶ Code

```
function test_moveFundsWhenMigrating2() public {
        joinGang(address(0));
        uint256 godFatherStartingBal = usdc.balanceOf(godFather);
        vm.startPrank(godFather);

        usdc.approve(address(moneyShelf), 500e6);
        laundrette.depositTheCrimeMoneyInATM(godFather, godFather, 250e6);
        laundrette.depositTheCrimeMoneyInATM(godFather, godFather, 250e6);
        assertEq(usdc.balanceOf(godFather), godFatherStartingBal - 500e6);
        assertEq(usdc.balanceOf(address(moneyShelf)), 500e6);
        assertEq(crimeMoney.balanceOf(godFather), 500e6);
        vm.stopPrank();

        EmergencyMigration migration = new EmergencyMigration();
        MoneyVault moneyVault = migration.migrate(kernel, usdc,
crimeMoney);


assertEq(address(kernel.getModuleForKeycode(Keycode.wrap("MONEY"))),
address(moneyVault));
        assertNotEq(usdc.balanceOf(address(moneyVault)), 500e6);

        vm.startPrank(godFather);
        laundrette.withdrawMoney(godFather, godFather, 500e6);
        vm.stopPrank();

        assertEq(usdc.balanceOf(godFather), godFatherStartingBal);
    }
```

Recommended Mitigation: Could use a boolean variable that keeps track of whether the module `MoneyShelf` has been upgraded or is active and when it has been upgraded it will disable deposits and withdrawals from the `MoneyShelf` contract.

Low

[L-1] Unchecked transfers

Description: The return value of an external transfer/transferFrom call is not checked

Impact: Several tokens do not revert in case of failure and return false. If the transfer fails, an attacker can call deposit for free or users can lose their tokens if transfer fails in withdraw.

While the token used (USDC) reverts on transfer failed, it is behind a proxy so in the future it can be upgraded to not revert on transfer.

it is aslo a good practice to check the return value of an external transfer/transferFrom call.

- Found in src/modules/MoneyShelf.sol Line: 27

```
        usdc.transferFrom(account, address(this), amount);
```

- Found in src/modules/MoneyShelf.sol Line: 34

```
        usdc.transfer(to, amount);
```

- Found in src/modules/MoneyVault.sol Line: 34

```
        usdc.transfer(to, amount);
```

Recommended Mitigation: It is recommended to use OpenZeppelin's SafeERC20 library.

[L-2] Missing checks for `address(0)` when assigning values to address state variables

Check for `address(0)` when assigning values to address state variables.

- Found in src/CrimeMoney.sol Line: 11

```
        kernel = _kernel;
```

- Found in src/Kernel.sol Line: 65

```
        kernel = kernel_;
```

- Found in src/Kernel.sol Line: 74

```
        kernel = newKernel_;
```

- Found in src/Kernel.sol Line: 209

```
            executor = target_;
```

- Found in src/Kernel.sol Line: 211

```
            admin = target_;
```

- Found in src/Kernel.sol Line: 226

```
        getModuleForKeycode[keycode] = newModule_;
```

- Found in src/Kernel.sol Line: 346

```
            modulePermissions[request.keycode][policy_]
[request.funcSelector] = grant_;
```

- Found in src/modules/MoneyShelf.sol Line: 16

```
        usdc = _usdc;
```

- Found in src/modules/MoneyShelf.sol Line: 17

```
        crimeMoney = _crimeMoney;
```

- Found in src/modules/MoneyVault.sol Line: 18

```
        usdc = _usdc;
```

- Found in src/modules/MoneyVault.sol Line: 19

```
        crimeMoney = _crimeMoney;
```

- Found in src/modules/Shelf.sol Line: 13

```
        bank[account] += amount;
```

- Found in src/modules/Shelf.sol Line: 17

```
        bank[account] -= amount;
```

[L-3] `public` functions not used internally could be marked `external`

Instead of marking a function as `public`, consider marking it as `external` if it is not used internally.

- Found in src/CrimeMoney.sol Line: 19

    ```
        function mint(address to, uint256 amount) public onlyMoneyShelf {
    ```

- Found in src/CrimeMoney.sol Line: 23

    ```
        function burn(address from, uint256 amount) public onlyMoneyShelf {
    ```

- Found in src/Kernel.sol Line: 381

    ```
        function grantRole(Role role_, address addr_) public onlyAdmin {
    ```

- Found in src/Kernel.sol Line: 392

    ```
        function revokeRole(Role role_, address addr_) public onlyAdmin {
    ```

- Found in src/modules/MoneyShelf.sol Line: 20

    ```
        function KEYCODE() public pure override returns (Keycode) {
    ```

- Found in src/modules/MoneyVault.sol Line: 22

    ```
        function KEYCODE() public pure override returns (Keycode) {
    ```

- Found in src/modules/WeaponShelf.sol Line: 10

    ```
        function KEYCODE() public pure override returns (Keycode) {
    ```

[L-4]: PUSH0 is not supported by all chains

Solc compiler version 0.8.20 switches the default target EVM version to Shanghai, which means that the generated bytecode will include PUSH0 opcodes. Be sure to select the appropriate EVM version in case you intend to deploy on a chain other than mainnet like L2 chains that may not support PUSH0, otherwise deployment of your contracts will fail.

- Found in src/Kernel.sol Line: 2

  ```
  pragma solidity ^0.8.15;
  ```

- Found in src/utils/KernelUtils.sol Line: 2

  ```
  pragma solidity ^0.8.15;
  ```

[L-5]: Modifiers invoked only once can be shoe-horned into the function

- Found in src/Kernel.sol Line: 177

  ```
      modifier onlyExecutor() {
  ```