Mafia Takedown Audit Report

Version 1.0

Ardeshir Gholami

June 14, 2024

Mafia Takedown Audit Report

Ardeshir Gholami

14 June 2024

Prepared by: Ardeshir Lead Auditors: - 4rdii

Table of Contents

Protocol Summary

An undercover AMA agent (anti-mafia agency) discovered a protocol used by the Mafia. In several days, a raid will be conducted by the police and we need as much information as possible about this protocol to prevent any problems. But the AMA doesn't have any web3 experts on their team. Hawks, they need your help! Find flaws in this protocol and send us your findings.

Disclaimer

I make 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
	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

```bash
--- script
--------- Deployer.s.sol
~~~

```
---------- EmergencyMigration.s.sol
--- src
---------- CrimeMoney.sol
---------- modules
---------- MoneyShelf.sol
---------- MoneyVault.sol
---------- Shelf.sol
---------- WeaponShelf.sol
--- policies
---------- Laundrette.sol
```

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. Known Issues # Executive Summary

Issues found

Severity	number of issues found
High	1
Medium	2
Low	2
Info	0
Total	5

Findings

High Risk Findings:

[H-1] Gang Members Can Still Withdraw In Emergency Mode

Description: When `configureDependencies` is called via `Kernel::_activatePolicy`, it then pushes the returned dependencies into a `Kernel::moduleDependents` mapping. This mapping is used to reconfigure the policy in case any of the modules which policy is dependent on changes.

However, in the `configureDependencies` function, when setting dependencies array to return the `MONEY` keycode is overwritten by `WEAPN` keycode.

```
function configureDependencies() external override onlyKernel returns (
    Keycode[] memory dependencies) {
        dependencies = new Keycode[](2);

        dependencies[0] = toKeycode("MONEY");
        moneyShelf = MoneyShelf(getModuleAddress(toKeycode("MONEY")));

@>      dependencies[0] = toKeycode("WEAPN");
        weaponShelf = WeaponShelf(getModuleAddress(toKeycode("WEAPN")))
            ;
    }
```

Impact: This does not affect the functionality of the contracts in normal state. However, when transitioning to emergency mode, the emergency deploy function executes `UpgradeModule` Action, which requires correct dependencies to work. Since the `Laundrette` policy is not set as a dependency for the `MONEY` keycode, upgrading this Keycode won't trigger this policy to reconfigure. As a result, all functions in `Laundrette` still work as if it is still in normal mode, and gang members still can call `WithdrawMoney` successfully.

Proof of Concept: To prove the concept, add the following test to `EmergencyMigration.t.sol`, Exploit Steps: 1. `gangM1` joins gang. 2. `gangM1` deposits some USDC to test it after emergency. 3. `godfather` enters emergency mode via calling migrate. 4. `gangM1` withdraws as if in normal mode. PoC:

```
function testGangMembersCanWithdrawInEmergency() public {
        //gangM1 joins gang
        address gangM1 = makeAddr("M1");
        joinGang(gangM1);
        vm.prank(godFather);
        usdc.transfer(gangM1, 100e6);
        //some deposits for contract to test it after emergency
        vm.startPrank(gangM1);
        usdc.approve(address(moneyShelf), 100e6);
        laundrette.depositTheCrimeMoneyInATM(gangM1, gangM1, 100e6);
        vm.stopPrank();
        assertEq(usdc.balanceOf(gangM1), 0);
        assertEq(usdc.balanceOf(address(moneyShelf)), 100e6);
        assertEq(address(kernel.getModuleForKeycode(Keycode.wrap("MONEY
            "))), address(moneyShelf));
        //enter emergency mode:

        EmergencyMigration migration = new EmergencyMigration();
        MoneyVault moneyVault = migration.migrate(kernel, usdc,
            crimeMoney);
        assertNotEq(address(moneyShelf), address(moneyVault));
        assertEq(address(kernel.getModuleForKeycode(Keycode.wrap("MONEY"))), address(moneyVault));

        //gangM1 withdraws in emergency mode:
        vm.startPrank(gangM1);
        laundrette.withdrawMoney(gangM1, gangM1, 100e6);
        assertEq(usdc.balanceOf(gangM1), 100e6);
        assertEq(usdc.balanceOf(address(moneyShelf)), 0);
    }
```

Recommended Mitigation: The fix is simple, just ensure the `configureDependencies` function returns the correct dependencies array:

```
function configureDependencies() external override onlyKernel returns (Keycode[] memory dependencies) {
        dependencies = new Keycode[](2);

        dependencies[0] = toKeycode("MONEY");
        moneyShelf = MoneyShelf(getModuleAddress(toKeycode("MONEY")));

-       dependencies[0] = toKeycode("WEAPN");
+       dependencies[1] = toKeycode("WEAPN");
        weaponShelf = WeaponShelf(getModuleAddress(toKeycode("WEAPN")));
    }
```

Meduim Risk Findings:

[M-1] Deposited USDC Stuck in `MoneyShelf` in Emergency Mode

Description: Note: This only happens if the bug in the `Laundrette::configureDependencies` is fixed.

```
    function configureDependencies() external override onlyKernel returns (Keycode[] memory dependencies) {
        dependencies = new Keycode[](2);

        dependencies[0] = toKeycode("MONEY");
        moneyShelf = MoneyShelf(getModuleAddress(toKeycode("MONEY")));

@>      dependencies[1] = toKeycode("WEAPN"); // this is the previous bug we assume is corrected.
        weaponShelf = WeaponShelf(getModuleAddress(toKeycode("WEAPN")));
    }
```

Upon addressing the bug in `configureDependencies`, a new issue arises where the `MoneyShelf` module fails to transfer deposited USDC to the newly upgraded `MoneyVault` during emergency mode. Additionally, the `bank` mapping is not updated accordingly, leaving the funds trapped within `MoneyShelf`.

Impact: Although this issue does not lead to permanent fund loss, and is fixable by just executing `UpgradeModule` and replacing the `MoneyVault` with `MoneyShelf` again. it severely impacts the functionality of Emergency Mode, rendering funds inaccessible to all actors involved.

Proof of Concept: To prove the concept, add the following test to `EmergencyMigration.t.sol`, Exploit Steps: 1. `gangM1` joins gang. 2. `gangM1` deposits some USDC to test it after emergency. 3. `godfather` enters emergency mode via calling migrate. 4. `gangM1` Can Not withdraw because in emergency mode and previous bug is fixed. 5. `godfather` Can Not withdraw because the funds are in `Moneyshell` and the `bank` mapping is not updated.

```
function testMoenyIsStuckInMoneyShelf() public {
        //gangM1 joins gang
        address gangM1 = makeAddr("M1");
        joinGang(gangM1);
        vm.prank(godFather);
        usdc.transfer(gangM1, 100e6);
        //some deposits for contract to test it after emergency
        vm.startPrank(gangM1);
        usdc.approve(address(moneyShelf), 100e6);
        laundrette.depositTheCrimeMoneyInATM(gangM1, gangM1, 100e6);
        vm.stopPrank();
        assertEq(usdc.balanceOf(gangM1), 0);
        assertEq(usdc.balanceOf(address(moneyShelf)), 100e6);
        assertEq(address(kernel.getModuleForKeycode(Keycode.wrap("MONEY
            "))), address(moneyShelf));
        //enter emergency mode:

        EmergencyMigration migration = new EmergencyMigration();
        MoneyVault moneyVault = migration.migrate(kernel, usdc,
            crimeMoney);
        assertNotEq(address(moneyShelf), address(moneyVault));
        assertEq(address(kernel.getModuleForKeycode(Keycode.wrap("MONEY
            "))), address(moneyVault));

        //gangM1 Can not withdraw in emergency mode:
        vm.startPrank(gangM1);
        vm.expectRevert("MoneyVault: only GodFather can receive USDC");
        laundrette.withdrawMoney(gangM1, gangM1, 100e6);

        //godfather Can NOT withdraw in emergency mode:
        uint256 gfBalanceBefore = usdc.balanceOf(godFather);

        vm.startPrank(godFather);
        vm.expectRevert(); // will revert with: arithmetic underflow or
             overflow (0x11) because Shelf.withdraw()
        laundrette.withdrawMoney(gangM1, godFather, 100e6);
        assertEq(usdc.balanceOf(godFather) - gfBalanceBefore, 0); // godFather balance change is zero
        assertEq(usdc.balanceOf(address(moneyShelf)), 100e6); // funds are still in moneyShelf
        assertEq(usdc.balanceOf(address(moneyVault)), 0); // moenyvault is empty
    }
```

Recommended Mitigation: Introduce an `emergencyMode` function within `MoneyShelf`. This function should facilitate the transfer of funds to the new `MoneyVault` whenever there's a change in the `MONEY` keycode. Modify `Laundrette` to invoke this function when necessary. Furthermore, directly update the `bank` mapping within `MoneyVault::withdrawUSDC` to circumvent the issue.

Recommended Mitigation Steps

First, implement the `emergencyMode` function in `MoneyShelf`:

```
+   function emergencyMode(address _newVault) external permissioned {
+       usdc.transfer(_newVault, usdc.balanceOf(address(this)));
+   }
```

Next, adjust `Laundrette::configureDependencies` and `Laundrette::requestPermissions` to utilize this new function:

```
function configureDependencies() external override onlyKernel returns (Keycode[] memory dependencies) {
        dependencies = new Keycode[](2);

        dependencies[0] = toKeycode("MONEY");
+       if (address(moneyShelf) != address(0)) {
+           address MoneyVault = getModuleAddress(toKeycode("MONEY"));
+           moneyShelf.emergencyMode(MoneyVault);
+       }
        moneyShelf = MoneyShelf(getModuleAddress(toKeycode("MONEY")));

        dependencies[1] = toKeycode("WEAPN");
        weaponShelf = WeaponShelf(getModuleAddress(toKeycode("WEAPN")));
    }

    function requestPermissions() external view override onlyKernel returns (Permissions[] memory requests) {
-       requests = new Permissions[](4);
+       requests = new Permissions[](5);

        requests[0] = Permissions(toKeycode("MONEY"), moneyShelf.depositUSDC.selector);
        requests[1] = Permissions(toKeycode("MONEY"), moneyShelf.withdrawUSDC.selector);
```

```

         requests[2] = Permissions(toKeycode("WEAPN"), weaponShelf.deposit.selector);
         requests[3] = Permissions(toKeycode("WEAPN"), weaponShelf.withdraw.selector);
+        requests[4] = Permissions(toKeycode("MONEY"), moneyShelf.emergencyMode.selector);

     }
```

Finally, modify `MoneyVault::withdrawUSDC` to bypass the need for updating the `bank` mapping and burning tokens through `crimeMoney`:

```
     function withdrawUSDC(address account, address to, uint256 amount) external {
         require(to == kernel.executor(), "MoneyVault: only GodFather can receive USDC");
-        withdraw(account, amount);
-        crimeMoney.burn(account, amount);
         usdc.transfer(to, amount);
     }
```

These adjustments ensure that the `GodFather` remains the sole authority for withdrawing funds during emergencies, but it will cause the loss of monetary value associated with `crimeMoney`.

[M-2] `MoneyVault` does not have `moneyshelf` role and as a result cant call `CrimeMoney::burn` or `CrimeMoney::mint`

Description: In the `Depoloyer.s.sol` script the moneyShelf contract is given the `moneyshelf` role, giving it access to `mint` or `burn` CrimeMoney tokens, but when the emergency mode happens (on top of previous bugs: incorrect dependencies, and funds being stuck in moneyshelf) the moneyVault doesnt have the `moneyshelf` role and can not call `burn` to withdraw tokens for `godfather`.

Impact: Funds remain inaccessible within the `moneyVault` due to the lack of permissions to withdraw them. Although a solution exists—assigning the `moneyshelf` role to `MoneyVault`—this issue underscores the importance of proper role management during emergency transitions.

Proof of Concept: Include the following test in migration tests, noting that previous bugs must be resolved before running this test.

```
     function testCantCallBurnLacksRole() public {
         //gangM1 joins gang
         address gangM1 = makeAddr("M1");
         joinGang(gangM1);
         vm.prank(godFather);
         usdc.transfer(gangM1, 100e6);
```

```
        //some deposits for contract to test it after emergency
        vm.startPrank(gangM1);
        usdc.approve(address(moneyShelf), 100e6);
        laundrette.depositTheCrimeMoneyInATM(gangM1, gangM1, 100e6);
        vm.stopPrank();
        assertEq(usdc.balanceOf(gangM1), 0);
        assertEq(usdc.balanceOf(address(moneyShelf)), 100e6);
        assertEq(address(kernel.getModuleForKeycode(Keycode.wrap("MONEY"))), address(moneyShelf));
        //enter emergency mode:

        EmergencyMigration migration = new EmergencyMigration();
        MoneyVault moneyVault = migration.migrate(kernel, usdc, crimeMoney);
        assertNotEq(address(moneyShelf), address(moneyVault));
        assertEq(address(kernel.getModuleForKeycode(Keycode.wrap("MONEY"))), address(moneyVault));

        //godfather Can NOT withdraw in emergency mode:
        uint256 gfBalanceBefore = usdc.balanceOf(godFather);

        vm.startPrank(godFather);
        vm.expectRevert("CrimeMoney: only MoneyShelf can mint");
        laundrette.withdrawMoney(gangM1, godFather, 100e6);
        assertEq(usdc.balanceOf(godFather) - gfBalanceBefore, 0); // godFather balance change is zero
        assertEq(usdc.balanceOf(address(moneyVault)), 100e6); // moenyvault is empty
    }
```

Recommended Mitigation: Assign the `moneyshelf` role to `MoneyVault` within the `EmergencyMigration` deployment script to enable token operations during emergency mode.

Low Risk Findings:

[L-1] Current Deployment Prevents Godfather from Adding Members to Gang

Description: Upon deployment, the `godfather` lacks the `gangmember` role, hindering their ability to use `Laundrette::addToTheGang` to add new members to the gang.

```
@>  function addToTheGang(address account) external onlyRole("gangmember") isGodFather {
        kernel.grantRole(Role.wrap("gangmember"), account);
    }
```

Impact: While the `godfather` can eventually acquire the `gangmember` role by first becoming the admin, granting themselves the role, and then reverting to the `Laundrette` admin, this process is a hassle. Removing the `gangmember` access control or assigning it to the `godfather` during deployment would streamline operations.

Proof of Concept: Include this test in `Laundrette.t.sol` to demonstrate the issue:

```
    function test_addToGang() public {
        vm.prank(godFather);
        vm.expectRevert(abi.encodeWithSelector(Policy_OnlyRole.selector
            , Role.wrap("gangmember")));
        laundrette.addToTheGang(address(123));
        ////walk around fix:
        vm.startPrank(godFather);
        kernel.executeAction(Actions.ChangeAdmin, godFather);
        kernel.grantRole(Role.wrap("gangmember"), godFather);
        kernel.executeAction(Actions.ChangeAdmin, address(laundrette));
        laundrette.addToTheGang(address(123));
        assertEq(kernel.hasRole(address(123), Role.wrap("gangmember")),
            true);
    }
```

Recommended Mitigation: To resolve this, consider removing the `onlyRole("gangmember")` access control since the `isGodFather` check is sufficient. Alternatively, if both checks are deemed necessary, assign the `gangmember` role to the `godfather` during deployment.

Fix 1: Remove the `onlyRole("gangmember")` requirement.

```
-    function addToTheGang(address account) external onlyRole("
    gangmember") isGodFather {
+    function addToTheGang(address account) external  isGodFather {

         kernel.grantRole(Role.wrap("gangmember"), account);
     }
```

Fix 2: Assign the `gangmember` role to the `godfather` during deployment.

```
     function deploy() public returns (Kernel, IERC20, CrimeMoney,
        WeaponShelf, MoneyShelf, Laundrette) {
         godFather = msg.sender;
         .
         .
         .

         kernel.grantRole(Role.wrap("moneyshelf"), address(moneyShelf));
+        kernel.grantRole(Role.wrap("gangmember"), godFather);

         .
         .
         .
     }
```

[L-2] `Laundrette::retrieveAdmin` Does Not Work Cause Only Executor Can Call `Kernel::executeAction`

Description: The `retrieveAdmin` function aims to allow the `godfather` to change the kernel admin, potentially terminating the `Laundrette` contract. However, this function is ineffective because it relies on `Laundrette` having the ability to call `Kernel::executeAction`, which is restricted to `kernel.executor()` alone. `Laundrette::retrieveAdmin`:

```
    function retrieveAdmin() external {
@>      kernel.executeAction(Actions.ChangeAdmin, kernel.executor());
    }
```

`Kernel::executeAction`:

```
@>  function executeAction(Actions action_, address target_) external
    onlyExecutor {
            .
            .
            .
    }
```

Impact: Despite being unusable due to the restriction (`Kernel_OnlyExecutor()`), this limitation is manageable because the `godfather` already possesses the `executor` role, enabling them to bypass this function and directly call `executeAction` on `Kernel`.

Proof of Concept: Include this test in `Laundrette.t.sol` to demonstrate the issue:

```
    function test_retrieveAdmin() public {
        vm.prank(godFather);
        vm.expectRevert(abi.encodeWithSelector(Kernel_OnlyExecutor.
            selector, address(laundrette)));
        laundrette.retrieveAdmin();

        // instead godfather can call this to become admin
        vm.prank(godFather);
        kernel.executeAction(Actions.ChangeAdmin, godFather);
        assertEq(kernel.admin(), godFather);
    }
```

Recommended Mitigation: Given that the `godfather` already holds the `executor` role, the `retrieveAdmin` function becomes redundant. Its removal simplifies the contract's interface without compromising functionality.

```
-    function retrieveAdmin() external {
-       kernel.executeAction(Actions.ChangeAdmin, kernel.executor());
-    }
```