Mafia Takedown Audit Report

Version 1.0

Keyword

June 8, 2024

Mafia Takedown Audit Report

xKeywordx

June 8th, 2024

Prepared by: xKeywordx

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.

Hawkers, they need your help!

Find flaws in this protocol and send us your findings.

This project uses the Default framework: https://github.com/fullyallocated/Default

Disclaimer

I make all efforts to find as many vulnerabilities in the code in the given time period, but hold no responsibilities for the findings provided in this document. A security audit 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

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

Scope

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

```
13         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.

Executive Summary

Known issues

Missing events. The Mafia knows that nothing is private on blockchains, view functions will reveal what the mafia owns. Users can deposit on any account, not only gang member's accounts.

Issues found

Severity	Number of issues found
High	2
Medium	1
Low	1
Gas	0
Info	0
Total	4

Findings

High

[H-1] Misconfigured dependencies in `Laundrette` contract can lead to potential vulnerabilities during a protocol upgrade

Description: The `Laundrette::configureDependencies` function incorrectly assigns the dependencies for the `MONEY` and `WEAPN` keycodes. The function assigns both keycodes to the same index in the dependencies array, causing the `MONEY` keycode to be overwritten by the `WEAPN` keycode. As a result, only `WEAPN` is correctly registered as a dependency in the `Kernel` contract.

Impact: Future `Policies` or `Modules` that rely on the correct registration of the `MONEY` keycode may encounter failures in permission assignments or role-based access control. Upgradability and maintenance risks. As the protocol evolves and `Modules` are upgraded, the `Kernel`'s dependency management may become inconsistent, leading to complex bugs.

Proof of Concepts: Add the following lines of code in the `Laundrette.t.sol` file.

PoC - click the arrow below

```
// change imports
import { Kernel, Policy, Permissions, Keycode, Role, Actions } from "src/Kernel.sol";

function testIncorrectDependencies() public {
        // Check the dependent index for the MONEY keycode
        Keycode moneyKeycode = Keycode.wrap(bytes5("MONEY"));
        uint256 moneyIndex = kernel.getDependentIndex(moneyKeycode, laundrette);
        console.log("Dependent Index for MONEY:", moneyIndex);

        // Check the dependent index for the WEAPN keycode
        Keycode weapnKeycode = Keycode.wrap(bytes5("WEAPN"));
        uint256 weapnIndex = kernel.getDependentIndex(weapnKeycode, laundrette);
        console.log("Dependent Index for WEAPN:", weapnIndex);

        // Verify the module dependents for each keycode
        uint256 moneyDependentsLength = getModuleDependentsLength(moneyKeycode);
        uint256 weapnDependentsLength = getModuleDependentsLength(weapnKeycode);

        console.log("Number of dependents for MONEY keycode:", moneyDependentsLength);
        console.log("Number of dependents for WEAPN keycode:", weapnDependentsLength);
```

```
21
22          // Ensure Laundrette is in the dependents list for both
                keycodes
23          bool foundInMoney = isDependentPresent(moneyKeycode, laundrette
                );
24          bool foundInWeapn = isDependentPresent(weapnKeycode, laundrette
                );
25
26          console.log("Laundrette found in MONEY dependents:",
                foundInMoney);
27          console.log("Laundrette found in WEAPN dependents:",
                foundInWeapn);
28
29          assertTrue(foundInMoney, "Laundrette should be a dependent of
                MONEY");
30          assertTrue(foundInWeapn, "Laundrette should be a dependent of
                WEAPN");
31      }
32
33      function getModuleDependentsLength(Keycode keycode) internal view
            returns (uint256) {
34          uint256 count = 0;
35          for (uint256 i = 0;; i++) {
36              try kernel.moduleDependents(keycode, i) returns (Policy) {
37                  count++;
38              } catch {
39                  break;
40              }
41          }
42          return count;
43      }
44
45      function isDependentPresent(Keycode keycode, Policy policy)
            internal view returns (bool) {
46          for (uint256 i = 0;; i++) {
47              try kernel.moduleDependents(keycode, i) returns (Policy
                    dependent) {
48                  if (dependent == policy) {
49                      return true;
50                  }
51              } catch {
52                  break;
53              }
54          }
55          return false;
56      }
```

Test output

```
1  Logs:
2    You have deployed a mock conract!
```

```
3    Make sure this was intentional
4    Dependent Index for MONEY: 0
5    Dependent Index for WEAPN: 0
6    Number of dependents for MONEY keycode: 0
7    Number of dependents for WEAPN keycode: 1
8    Laundrette found in MONEY dependents: false
9    Laundrette found in WEAPN dependents: true
```

Recommended mitigation: Update the `Laundrette::configureDependencies` function.

```
1      function configureDependencies() external override onlyKernel
           returns (Keycode[] memory dependencies) {
2          dependencies = new Keycode[](2);
3
4          dependencies[0] = toKeycode("MONEY");
5          moneyShelf = MoneyShelf(getModuleAddress(toKeycode("MONEY")));
6
7  -       dependencies[0] = toKeycode("WEAPN");
8  +       dependencies[1] = toKeycode("WEAPN");
9          weaponShelf = WeaponShelf(getModuleAddress(toKeycode("WEAPN")))
               ;
10     }
```

[H-2] An attacker can steal the funds of other users by depositing their tokens into the protocol on behalf of his account, if the victim approved the `moneyShelf` contract for deposits.

Description: The `Laundrette::depositTheCrimeMoneyInATM` function takes two address parameters as input, an `account` which is the address that we will get the tokens `from` and an address `to` which is the address on behalf of which tokens are deposited into the protocol. The `Shelf::deposit` function will credit the amount of tokens deposited on behalf of `address to` when someone calls the `depositTheCrimeMoneyInATM` function.

The problem arises from the fact that if user A approves the protocol to spend 1000 USDC, user B can call the `Laundrette::depositTheCrimeMoneyInATM` function with `account == user A's address`, `to == user B's address`, `amount == 1000 USDC`.

Impact: User B is able to steal the funds of user A, by depositing user A's tokens into the protocol, but getting the amount credited to his address instead.

Proof of Concepts: Place the following test into the `Laundrette.t.sol` file.

PoC - click the arrow below

```
1  // change imports
2  import { Kernel, Policy, Permissions, Keycode, Role, Actions } from "
       src/Kernel.sol";
3
```

```
    function test_Deposit() public {
        //Create 2 users
        address alice = makeAddr("alice");
        address bob = makeAddr("bob");

        // Send them some USDC
        vm.startPrank(godFather);
        usdc.transfer(alice, 100e6);
        usdc.transfer(bob, 100e6);
        vm.stopPrank();

        assertEq(usdc.balanceOf(address(alice)), 100e6);
        assertEq(usdc.balanceOf(address(bob)), 100e6);

        // Approve USDC for deposit
        vm.prank(alice);
        usdc.approve(address(moneyShelf), 100e6);
        vm.prank(bob);
        usdc.approve(address(moneyShelf), 100e6);

        // Bob deposits Alice's tokens on his behalf
        laundrette.depositTheCrimeMoneyInATM(alice, bob, 100e6);
        laundrette.depositTheCrimeMoneyInATM(bob, bob, 100e6);

        // Assert
        assertEq(usdc.balanceOf(address(alice)), 0);
        assertEq(usdc.balanceOf(address(bob)), 0);

        console.log(moneyShelf.getAccountAmount(bob));
        console.log(moneyShelf.getAccountAmount(alice));

        // Bob is added to the gang and withdraws the money
        vm.prank(kernel.admin());
        kernel.grantRole(Role.wrap("gangmember"), godFather);
        vm.prank(godFather);
        laundrette.addToTheGang(bob);
        vm.prank(bob);
        laundrette.withdrawMoney(bob, bob, 200e6);
        assertEq(usdc.balanceOf(address(bob)), 200e6);
    }
```

Test output

```
├
    [562] MockUSDC::balanceOf(bob: [0
        x1D96F2f6BeF1202E4Ce1Ff6Dad0c2CB002861d3e]) [staticcall]│└←
        [Return] 200000000 [2e8]├
    [0] VM::assertEq(200000000 [2e8], 200000000 [2e8]) [staticcall]
        │└←
        [Return]└←
```

```
6         [Stop]
7
8   Suite result: ok. 1 passed; 0 failed; 0 skipped; finished in 1.70ms
      (497.99µs CPU time)
```

Recommended mitigation: Add checks to enforce that the address depositing the tokens is the actual sender of the transaction.

```
1       function depositTheCrimeMoneyInATM(address account, address to,
            uint256 amount) external {
2   +       require(account == msg.sender, "Not token owner");
3           moneyShelf.depositUSDC(account, to, amount);
4       }
```

Medium

[M-1] Malicious gangmember can remove the `gangmember` role of other gangmembers, including the `godfather`'s address `gangmember` role.

Description: The `Laundrette::quitTheGang` function can be called by anyone and it remove the `gangmember` role of the `account`. A malicious gangmember can call this function to remove the `gangmember` role of other gangmembers, including the removel of the `gangmember` role for the `godfather` address.

Impact: A gangmember can go rogue and remove all his partners in crime from the gang.

Proof of Concepts: Put the code below in the `Laundrette.t.sol` file

PoC - click the arrow below

```
1   // change imports
2   import { Kernel, Policy, Permissions, Keycode, Role, Actions } from "
      src/Kernel.sol";
3
4       function test_quitTheGangGodfather() public {
5           vm.prank(kernel.admin());
6           kernel.grantRole(Role.wrap("gangmember"), godFather);
7           address alice = makeAddr("alice");
8           address bob = makeAddr("bob");
9           address michael = makeAddr("michael");
10          vm.startPrank(godFather);
11          laundrette.addToTheGang(address(alice));
12          laundrette.addToTheGang(address(bob));
13          laundrette.addToTheGang(address(michael));
14          vm.stopPrank();
15          vm.startPrank(alice);
16          laundrette.quitTheGang(godFather);
```

```
        laundrette.quitTheGang(bob);
        laundrette.quitTheGang(michael);
        vm.stopPrank();
        assertNotEq(kernel.hasRole(address(godFather), Role.wrap("
            gangmember")), true);
        assertNotEq(kernel.hasRole(address(bob), Role.wrap("gangmember"
            )), true);
        assertNotEq(kernel.hasRole(address(michael), Role.wrap("
            gangmember")), true);
    }
```

Test output

```
├─
    [707] Kernel::hasRole(God Father: [0
        xe166Ae83c3384a19498Ae0674706988DD2797489], 0
        x67616e676d656d6265720000000000000000000000000000000000000000000000
        ) [staticcall]│└─←
        [Return] false├─
    [0] VM::assertNotEq(false, true) [staticcall]│└─←
        [Return]├─
    [707] Kernel::hasRole(bob: [0
        x1D96F2f6BeF1202E4Ce1Ff6Dad0c2CB002861d3e], 0
        x67616e676d656d6265720000000000000000000000000000000000000000000000
        ) [staticcall]│└─←
        [Return] false├─
    [0] VM::assertNotEq(false, true) [staticcall]│└─←
        [Return]├─
    [707] Kernel::hasRole(michael: [0
        x45c8b98893D082f913896Eeca50AB00Db225298d], 0
        x67616e676d656d6265720000000000000000000000000000000000000000000000
        ) [staticcall]│└─←
        [Return] false├─
    [0] VM::assertNotEq(false, true) [staticcall]│└─←
        [Return]└─←
     [Stop]

Suite result: ok. 1 passed; 0 failed; 0 skipped; finished in 1.96ms
    (647.04µs CPU time)
```

Recommended mitigation: Use the `isGodFather` modifier on the `quitTheGang` function to make it permissioned or add a check that enforces that a gang member can only remove his own `gangmember` role.

```
    function quitTheGang(address account) external onlyRole("gangmember
        ") {
+       require(account == msg.sender, "Not so fast, traitor! You can
    only quit your own membership.");
```

```
3           kernel.revokeRole(Role.wrap("gangmember"), account);
4       }
```

Low

[L-1] `Laundrette::retrieveAdmin` function will always revert.

Description: The `Laundrette::retrieveAdmin` function is misconfigured. Instead of reading the value of the public `admin` address from the `Kernel` contract, the function calls the `Kernel::executeAction` function and attempts to change the `admin` address with the `executor`'s address. The `Kernel::executeAction` function is guarded by an `onlyExecutor` modifier. Because `msg.sender` in the context of the `Kernel` contract will be the `Laundrette` contract and not the `kernel.executor()` address, this function will always revert.

Impact: Users can not read the value of the `kernel.admin()` address by calling this function.

Proof of Concepts: Insert the code below in the `Laundrette.t.sol` file.

```
1       function test_retrieveAdmin() public {
2           address alice = makeAddr("alice");
3           vm.prank(alice);
4           vm.expectRevert();
5           laundrette.retrieveAdmin();
6       }
```

Recommended mitigation: Change the function by following the code sample below

```
1   -   function retrieveAdmin() external {
2   -       kernel.executeAction(Actions.ChangeAdmin, kernel.executor());
3   +   function retrieveAdmin() external view returns (address) {
4   +       return kernel.admin();
5       }
```