智能合约安全审计报告

[2021]

目录

1 前言

慢雾安全团队于 2021.09.13，收到 DODO 团队对 DODO-NFTPool 智能合约安全审计的申请，慢雾安全团队根据项目特点制定如下审计方案。

慢雾安全团队将采用“白盒为主，黑灰为辅”的策略，以最贴近真实攻击的方式，对项目进行安全审计。

慢雾科技项目测试方法：

测试方法	说明
黑盒测试	站在外部从攻击者角度进行安全测试。
灰盒测试	通过脚本工具对代码模块进行安全测试，观察内部运行状态，挖掘弱点。
白盒测试	基于项目的源代码，进行脆弱性分析和漏洞挖掘。

慢雾科技漏洞风险等级：

漏洞等级	说明
严重漏洞	严重漏洞会对项目的安全造成重大影响，强烈建议修复严重漏洞。
高危漏洞	高危漏洞会影响项目的正常运行，强烈建议修复高危漏洞。
中危漏洞	中危漏洞会影响项目的运行，建议修复中危漏洞。
低危漏洞	低危漏洞可能在特定场景中会影响项目的业务操作，建议项目方自行评估和考虑这些问题是否需要修复。
弱点	理论上存在安全隐患，但工程上极难复现。
增强建议	编码或架构存在更好的实践方法。

2 审计方法

慢雾安全团队智能合约安全审计流程包含两个步骤:

- 使用开源或内部自动化分析的工具对合约代码中常见的安全漏洞进行扫描和测试。
- 人工审计代码的安全问题，通过人工分析合约代码，发现代码中潜在的安全问题。

如下是合约代码审计过程中慢雾安全团队会重点审查的漏洞列表:

（其他未知的安全漏洞及审计项不包含在本次审计责任范围）

- 重入漏洞
- 重放漏洞
- 重排漏洞
- 短地址漏洞
- 拒绝服务漏洞
- 条件竞争漏洞
- 权限控制漏洞
- 整数上溢/下溢漏洞
- 时间戳依赖漏洞
- 未声明的存储指针漏洞
- 算术精度误差漏洞
- tx.origin身份验证漏洞
- 假充值漏洞
- 变量覆盖漏洞
- Gas优化审计
- 恶意 Event 事件审计
- 冗余的回调函数
- 不安全的外部调用审计
- 函数状态变量可见性审计

- 业务逻辑缺陷审计
- 变量声明及作用域审计

3 项目概要

3.1 项目介绍

审计版本代码：

https://github.com/DODOEX/contractV2/tree/feature/nftPool

commit: e16ceb038ed6bf070ea75d9359c7ad54e6f3e226

审计范围内的文件：

- contracts/NFTPool/impl/FilterAdmin.sol
- contracts/NFTPool/impl/BaseFilterV1.sol
- contracts/NFTPool/impl/FilterERC721V1.sol
- contracts/NFTPool/impl/FilterERC1155V1.sol
- contracts/NFTPool/impl/Controller.sol
- contracts/SmartRoute/proxies/DODONFTPoolProxy.sol
- contracts/SmartRoute/DODONFTApprove.sol

修复版本代码：

https://github.com/DODOEX/contractV2/tree/feature/nftPool

commit: 7b526b8208ec172b4ccfabbf22b39edcbe284561

3.2 漏洞信息

如下是本次审计发现的漏洞及漏洞的修复状态信息：

NO	标题	漏洞类型	漏洞等级	漏洞状态
N1	初始化操作安全提醒	条件竞争攻击	建议	已确认
N2	SuperOwner 权限过大问题	权限控制攻击	中	已确认
N3	Owner 权限过大问题	权限控制攻击	高	已确认
N4	ALLOWED_PROXY 权限过大问题	权限控制攻击	高	已确认
N5	随机数可被预测问题	区块数据依赖攻击	高	已确认
N6	业务逻辑错误	业务逻辑缺陷审计	高	已修复
N7	同名函数的安全提醒	其它	建议	已确认
N8	算术运算没有使用 SafeMath	整数上溢/下溢攻击	低	已修复
N9	函数可见性问题	Gas优化设计	低	已修复
N10	开放式外部调用	不安全的外部调用	严重	已修复

4 审计详情

4.1 合约基础信息

如下是合约主网地址：

目前代码还未部署到主网。

4.2 函数可见性分析

在审计过程中，慢雾安全团队对核心合约的函数可见性进行分析，结果如下：

BaseFilterV1			
Function Name	Visibility	Mutability	Modifiers
isNFTValid	External	-	-
isNFTIDValid	Public	-	-
getAvaliableNFTInAmount	Public	-	-
getAvaliableNFTOutAmount	Public	-	-
getNFTIndexById	Public	-	-
queryNFTIn	Public	-	-
queryNFTTargetOut	Public	-	-
queryNFTRandomOut	Public	-	-
_geometricCalc	Internal	-	-
_getRandomNum	Public	-	-
changeNFTInPrice	External	Can Modify State	onlySuperOwner
_changeNFTInPrice	Internal	Can Modify State	-
changeNFTRandomOutPrice	External	Can Modify State	onlySuperOwner
_changeNFTRandomOutPrice	Internal	Can Modify State	-
changeNFTTargetOutPrice	External	Can Modify State	onlySuperOwner
_changeNFTTargetOutPrice	Internal	Can Modify State	-
changeNFTAmountRange	External	Can Modify State	onlySuperOwner
_changeNFTAmountRange	Internal	Can Modify State	-
changeTokenIdRange	External	Can Modify State	onlySuperOwner

BaseFilterV1			
_changeTokenIdRange	Internal	Can Modify State	-
changeTokenIdMap	External	Can Modify State	onlySuperOwner
_changeTokenIdMap	Internal	Can Modify State	-
changeFilterName	External	Can Modify State	onlySuperOwner

FilterERC721V1			
Function Name	Visibility	Mutability	Modifiers
init	External	Can Modify State	-
ERC721In	External	Can Modify State	preventReentrant
ERC721TargetOut	External	Can Modify State	preventReentrant
ERC721RandomOut	External	Can Modify State	preventReentrant
onERC721Received	External	Can Modify State	-
_transferOutERC721	Internal	Can Modify State	-
emergencyWithdraw	External	Can Modify State	onlySuperOwner
supportsInterface	Public	-	-
version	External	-	-

FilterERC1155V1			
Function Name	Visibility	Mutability	Modifiers
init	External	Can Modify State	-
ERC1155In	External	Can Modify State	preventReentrant

FilterERC1155V1			
ERC1155TargetOut	External	Can Modify State	preventReentrant
ERC1155RandomOut	External	Can Modify State	preventReentrant
onERC1155Received	External	Can Modify State	-
onERC1155BatchReceived	External	Can Modify State	-
_transferOutERC1155	Internal	Can Modify State	-
emergencyWithdraw	External	Can Modify State	onlySuperOwner
_maintainERC1155Out	Internal	Can Modify State	-
_maintainERC1155In	Internal	Can Modify State	-
supportsInterface	Public	-	-
version	External	-	-

DODONFTPoolProxy			
Function Name	Visibility	Mutability	Modifiers
<Constructor>	Public	Can Modify State	-
erc721In	External	Can Modify State	-
erc721TargetOut	External	Can Modify State	-
erc721RandomOut	External	Can Modify State	-
erc1155In	External	Can Modify State	-
erc1155TargetOut	External	Can Modify State	-
erc1155RandomOut	External	Can Modify State	-

DODONFTPoolProxy			
createNewNFTPoolV1	External	Can Modify State	-
createFilterV1	Public	Can Modify State	-
erc721ToErc20	External	Can Modify State	preventReentrant
changeMaintainer	External	Can Modify State	onlyOwner
changeFilterAdminTemplate	External	Can Modify State	onlyOwner
changeController	External	Can Modify State	onlyOwner
setFilterTemplate	External	Can Modify State	onlyOwner
_generalApproveMax	Internal	Can Modify State	-
_generalBalanceOf	Internal	-	-
_generalTransfer	Internal	Can Modify State	-

DODONFTApprove			
Function Name	Visibility	Mutability	Modifiers
init	External	Can Modify State	-
unlockAddProxy	External	Can Modify State	onlyOwner
lockAddProxy	Public	Can Modify State	onlyOwner
addDODOProxy	External	Can Modify State	onlyOwner notLocked
removeDODOProxy	External	Can Modify State	onlyOwner
claimERC721	External	Can Modify State	-
claimERC1155	External	Can Modify State	-

DODONFTApprove			
claimERC1155Batch	External	Can Modify State	-
isAllowedProxy	External	-	-

FilterAdmin			
Function Name	Visibility	Mutability	Modifiers
init	External	Can Modify State	-
mintFragTo	External	Can Modify State	-
burnFragFrom	External	Can Modify State	-
queryMintFee	Public	Can Modify State	-
queryBurnFee	Public	Can Modify State	-
isRegisteredFilter	Public	-	-
getFilters	Public	-	-
addFilter	External	Can Modify State	onlyOwner
changeFeeRate	External	Can Modify State	onlyOwner
version	External	-	-

Controller			
Function Name	Visibility	Mutability	Modifiers
setFilterAdminFeeRateInfo	External	Can Modify State	onlyOwner
setGlobalParam	External	Can Modify State	onlyOwner
setEmergencyWithdraw	External	Can Modify State	onlyOwner

Controller			
getMintFeeRate	External	-	-
getBurnFeeRate	External	-	-

4.3 漏洞详情

[N1] [建议] 初始化操作安全提醒

漏洞类型: 条件竞争攻击

详细内容

通过调用 initOwner 进行初始化 `_OWNER_` 的操作，存在被恶意攻击者抢先调用 initOwner 函数进行初始化的问题。

- lib/InitializableOwnable.sol#L42-L45

```
    function initOwner(address newOwner) public notInitialized {
        _INITIALIZED_ = true;
        _OWNER_ = newOwner;
    }
```

- NFTPool/impl/FilterERC1155V1.sol#L27-L53

```
    function init(
        address filterAdmin,
        address nftCollection,
        bool[] memory toggles,
        string memory filterName,
        uint256[] memory numParams, //0 - startId, 1 - endId, 2 - maxAmount, 3 -
minAmount
        uint256[] memory priceRules,
        uint256[] memory spreadIds
    ) external {
        initOwner(filterAdmin);

        _FILTER_NAME_ = filterName;
        _NFT_COLLECTION_ = nftCollection;
```

```
        _changeNFTInPrice(priceRules[0], priceRules[1], toggles[0]);
        _changeNFTRandomOutPrice(priceRules[2], priceRules[3], toggles[1]);
        _changeNFTTargetOutPrice(priceRules[4], priceRules[5], toggles[2]);

        _changeNFTAmountRange(numParams[2], numParams[3]);

        _changeTokenIdRange(numParams[0], numParams[1]);
        for (uint256 i = 0; i < spreadIds.length; i++) {
            _SPREAD_IDS_REGISTRY_[spreadIds[i]] = true;
        }

        emit FilterInit(filterAdmin, nftCollection, filterName);
    }
```

- NFTPool/impl/FilterERC721V1.sol#L30-L55

```
function init(
        address filterAdmin,
        address nftCollection,
        bool[] memory toggles,
        string memory filterName,
        uint256[] memory numParams, //0 - startId, 1 - endId, 2 - maxAmount, 3 -
minAmount
        uint256[] memory priceRules,
        uint256[] memory spreadIds
    ) external {
        initOwner(filterAdmin);
        _FILTER_NAME_ = filterName;

        _NFT_COLLECTION_ = nftCollection;
        _changeNFTInPrice(priceRules[0], priceRules[1], toggles[0]);
        _changeNFTRandomOutPrice(priceRules[2], priceRules[3], toggles[1]);
        _changeNFTTargetOutPrice(priceRules[4], priceRules[5], toggles[2]);

        _changeNFTAmountRange(numParams[2], numParams[3]);

        _changeTokenIdRange(numParams[0], numParams[1]);
        for (uint256 i = 0; i < spreadIds.length; i++) {
            _SPREAD_IDS_REGISTRY_[spreadIds[i]] = true;
        }

        emit FilterInit(filterAdmin, nftCollection, filterName);
    }
```

解决方案

1. 建议 initOwner 和 init 操作可以在其他合约中创建合约后立即在同一笔交易内调用，避免被攻击者恶意调用。
2. 如果采用 porxy 和 implementation 的架构方式，也需要在实现合约中调用 initOwner 和 init 完成初始化操作。

漏洞状态

已确认

[N2] [中] SuperOwner 权限过大问题

漏洞类型: 权限控制攻击

详细内容

onlySuperOwner 角色可以修改合约中的配置，如: 修改 NFT 价格，修改上架的 NFT 信息等，且存在交易所顺序依赖的问题，如果 SuperOwner 抢先修改价格话，会让用户以非预期的价格进行购买。

- NFTPool/impl/BaseFilterV1.sol#L109-L303

```
function changeNFTInPrice(
        uint256 newGsStart,
        uint256 newCr,
        bool toggleFlag
    ) external onlySuperOwner {
        _changeNFTInPrice(newGsStart, newCr, toggleFlag);
    }

    function _changeNFTInPrice(
        uint256 newGsStart,
        uint256 newCr,
        bool toggleFlag
    ) internal {
        require(newCr > DecimalMath.ONE, "CR_INVALID");
        _GS_START_IN_ = newGsStart;
        _CR_IN_ = newCr;
        _NFT_IN_TOGGLE_ = true;

        emit ChangeNFTInPrice(newGsStart, newCr, toggleFlag);
    }
```

```
function changeNFTRandomOutPrice(
    uint256 newGsStart,
    uint256 newCr,
    bool toggleFlag
) external onlySuperOwner {
    _changeNFTRandomOutPrice(newGsStart, newCr, toggleFlag);
}

function _changeNFTRandomOutPrice(
    uint256 newGsStart,
    uint256 newCr,
    bool toggleFlag
) internal {
    require(newCr > DecimalMath.ONE, "CR_INVALID");
    _GS_START_RANDOM_OUT_ = newGsStart;
    _CR_RANDOM_OUT_ = newCr;
    _NFT_RANDOM_OUT_TOGGLE_ = true;

    emit ChangeNFTRandomOutPrice(newGsStart, newCr, toggleFlag);
}

function changeNFTTargetOutPrice(
    uint256 newGsStart,
    uint256 newCr,
    bool toggleFlag
) external onlySuperOwner {
    _changeNFTTargetOutPrice(newGsStart, newCr, toggleFlag);
}

function _changeNFTTargetOutPrice(
    uint256 newGsStart,
    uint256 newCr,
    bool toggleFlag
) internal {
    require(newCr > DecimalMath.ONE, "CR_INVALID");
    _GS_START_TARGET_OUT_ = newGsStart;
    _CR_TARGET_OUT_ = newCr;
    _NFT_TARGET_OUT_TOGGLE_ = true;

    emit ChangeNFTTargetOutPrice(newGsStart, newCr, toggleFlag);
}

function changeNFTAmountRange(uint256 maxNFTAmount, uint256 minNFTAmount)
    external
```

```
        onlySuperOwner
    {
        _changeNFTAmountRange(maxNFTAmount, minNFTAmount);
    }

    function _changeNFTAmountRange(uint256 maxNFTAmount, uint256 minNFTAmount)
internal {
        require(maxNFTAmount >= minNFTAmount, "AMOUNT_INVALID");
        _MAX_NFT_AMOUNT_ = maxNFTAmount;
        _MIN_NFT_AMOUNT_ = minNFTAmount;

        emit ChangeNFTAmountRange(maxNFTAmount, minNFTAmount);
    }

    function changeTokenIdRange(uint256 nftIdStart, uint256 nftIdEnd) external
onlySuperOwner {
        _changeTokenIdRange(nftIdStart, nftIdEnd);
    }

    function _changeTokenIdRange(uint256 nftIdStart, uint256 nftIdEnd) internal {
        require(nftIdStart <= nftIdEnd, "TOKEN_RANGE_INVALID");

        _NFT_ID_START_ = nftIdStart;
        _NFT_ID_END_ = nftIdEnd;

        emit ChangeTokenIdRange(nftIdStart, nftIdEnd);
    }

    function changeTokenIdMap(uint256[] memory tokenIds, bool[] memory isRegistered)
        external
        onlySuperOwner
    {
        _changeTokenIdMap(tokenIds, isRegistered);
    }

    function _changeTokenIdMap(uint256[] memory tokenIds, bool[] memory isRegistered)
internal {
        require(tokenIds.length == isRegistered.length, "PARAM_NOT_MATCH");

        for (uint256 i = 0; i < tokenIds.length; i++) {
            _SPREAD_IDS_REGISTRY_[tokenIds[i]] = isRegistered[i];
            emit ChangeTokenIdMap(tokenIds[i], isRegistered[i]);
        }
    }
```

```
function changeFilterName(string memory newFilterName)
    external
    onlySuperOwner
{
    _FILTER_NAME_ = newFilterName;
    emit ChangeFilterName(newFilterName);
}
```

解决方案

建议可以添加用户期望的成交的价格区间管理，避免因为 SuperOwner 提前修改价格导致最终成交的价格与用户的期望不一致。

漏洞状态

已确认；经过与项目的沟通讨论，这里 SuperOwner 是 NFT 的做市商，因为业务设计的需要所以 SuperOwner 要有这么大的权限。

[N3] [高] Owner 权限过大问题

漏洞类型: 权限控制攻击

详细内容

Owner 可以修改 Filter 合约的配置，如：设置开启紧急提款的开关等。

- NFTPool/impl/Controller.sol#L37-L63

```
function setFilterAdminFeeRateInfo(
      address filterAdminAddr,
      uint256 nftInFeeRate,
      uint256 nftOutFeeRate,
      bool isOpen
  ) external onlyOwner {
      FilterAdminFeeRateInfo memory feeRateInfo = FilterAdminFeeRateInfo({
          nftInFeeRate: nftInFeeRate,
          nftOutFeeRate: nftOutFeeRate,
          isOpen: isOpen
      });
      filterAdminFeeRates[filterAdminAddr] = feeRateInfo;

      emit SetFilterAdminFeeRateInfo(filterAdminAddr, nftInFeeRate, nftOutFeeRate,
```

```
isOpen);
    }

    function setGlobalParam(uint256 nftInFeeRate, uint256 nftOutFeeRate) external
onlyOwner {
        _GLOBAL_NFT_IN_FEE_RATE_ = nftInFeeRate;
        _GLOBAL_NFT_OUT_FEE_RATE_ = nftOutFeeRate;

        emit SetGlobalParam(nftInFeeRate, nftOutFeeRate);
    }

    function setEmergencyWithdraw(address filter, bool isOpen) external onlyOwner {
        isEmergencyWithdrawOpen[filter] = isOpen;
        emit SetEmergencyWithdraw(filter, isOpen);
    }
```

开启紧急提款开关的 Filter 合约，SuperOwner 可以通过 emergencyWithdraw 函数将合约中的资产提取出来。

- NFTPool/impl/FilterERC1155V1.sol#L142-L166

```
function emergencyWithdraw(
        address[] memory nftContract,
        uint256[] memory tokenIds,
        uint256[] memory amounts,
        address to
    ) external onlySuperOwner {
        require(
            nftContract.length == tokenIds.length && nftContract.length ==
amounts.length,
            "PARAM_INVALID"
        );
        address controller = IFilterAdmin(_OWNER_)._CONTROLLER_();
        require(
            IController(controller).isEmergencyWithdrawOpen(address(this)),
            "EMERGENCY_WITHDRAW_NOT_OPEN"
        );

        for (uint256 i = 0; i < nftContract.length; i++) {
            uint256 tokenId = tokenIds[i];
            IERC1155(nftContract[i]).safeTransferFrom(address(this), to, tokenId,
amounts[i], "");
            if (_NFT_RESERVE_[tokenId] > 0 && nftContract[i] == _NFT_COLLECTION_) {
                _maintainERC1155Out(tokenId);
            }
```

```
            emit EmergencyWithdraw(nftContract[i],tokenIds[i], amounts[i], to);
        }
    }
```

- NFTPool/impl/FilterERC721V1.sol#L137-L163

```
function emergencyWithdraw(
        address[] memory nftContract,
        uint256[] memory tokenIds,
        address to
    ) external onlySuperOwner {
        require(nftContract.length == tokenIds.length, "PARAM_INVALID");
        address controller = IFilterAdmin(_OWNER_)._CONTROLLER_();
        require(
            IController(controller).isEmergencyWithdrawOpen(address(this)),
            "EMERGENCY_WITHDRAW_NOT_OPEN"
        );

        for (uint256 i = 0; i < nftContract.length; i++) {
            uint256 tokenId = tokenIds[i];
            if (_NFT_RESERVE_[tokenId] > 0 && nftContract[i] == _NFT_COLLECTION_) {
                uint256 index = getNFTIndexById(tokenId);
                _NFT_IDS_[index] = _NFT_IDS_[_NFT_IDS_.length - 1];
                _NFT_IDS_.pop();
                _NFT_RESERVE_[tokenId] = 0;
                _TOKENID_IDX_[tokenId] = 0;
                _TOKENID_IDX_[_NFT_IDS_[index]] = index + 1;
            }
            IERC721(nftContract[i]).safeTransferFrom(address(this), to, tokenIds[i]);
            emit EmergencyWithdraw(nftContract[i],tokenIds[i],to);
        }
        _TOTAL_NFT_AMOUNT_ = _NFT_IDS_.length;
    }
```

解决方案

1. 建议将 Owner 角色的权限移交给 timelock 合约或治理合约。
2. 建议采用多签的策略避免因为私钥丢失导致合约配置被恶意更改。

漏洞状态

已确认

[N4] [高] ALLOWED_PROXY 权限过大问题

漏洞类型: 权限控制攻击

详细内容

ALLOWED_PROXY 角色可以将授权给合约的额度转出，存在权限过大的问题。

- NFTPool/impl/DODONFTApprove.sol#L70-L100

```
    function claimERC721(
        address nftContract,
        address who,
        address dest,
        uint256 tokenId
    ) external {
        require(_IS_ALLOWED_PROXY_[msg.sender], "DODONFTApprove:Access restricted");
        IERC721(nftContract).safeTransferFrom(who, dest, tokenId);
    }

    function claimERC1155(
        address nftContract,
        address who,
        address dest,
        uint256 tokenId,
        uint256 amount
    ) external {
        require(_IS_ALLOWED_PROXY_[msg.sender], "DODONFTApprove:Access restricted");
        IERC1155(nftContract).safeTransferFrom(who, dest, tokenId, amount, "");
    }

    function claimERC1155Batch(
        address nftContract,
        address who,
        address dest,
        uint256[] memory tokenIds,
        uint256[] memory amounts
    ) external {
        require(_IS_ALLOWED_PROXY_[msg.sender], "DODONFTApprove:Access restricted");
        IERC1155(nftContract).safeBatchTransferFrom(who, dest, tokenIds, amounts,
"");
    }
```

解决方案

建议要设置好授权额度的范围，按照需要进行设置。不能设置为最大值。

漏洞状态

已确认

[N5] [高] 随机数可被预测问题

漏洞类型: 区块数据依赖攻击

详细内容

项目中需要通过随机的方式获取 NFT，代码实现上采用链上的区块数据作为随机数的来源，存在随机数可被预测的问题，且采用了获取随机数的同时便确定 NFT 的编号，存在回滚攻击的问题。

- NFTPool/impl/BaseFilterV1.sol#L182-L186

```
    function _getRandomNum() public view returns (uint256 randomNum) {
        randomNum = uint256(
            keccak256(abi.encodePacked(tx.origin, blockhash(block.number - 1),
gasleft()))
        );
    }
```

- NFTPool/impl/FilterERC721V1.sol#L98-L112

```
 function ERC721RandomOut(uint256 amount, address to)
        external
        preventReentrant
        returns (uint256 paid)
    {
        (uint256 rawPay, ) = queryNFTRandomOut(amount);
        paid = IFilterAdmin(_OWNER_).burnFragFrom(msg.sender, rawPay);
        for (uint256 i = 0; i < amount; i++) {
            uint256 index = _getRandomNum() % _TOTAL_NFT_AMOUNT_;
            _transferOutERC721(to, _NFT_IDS_[index]);

            emit RandomOut(_NFT_IDS_[index]);
        }
```

```
        _TOTAL_NFT_AMOUNT_ = _NFT_IDS_.length;
    }
```

- NFTPool/impl/FilterERC1155V1.sol#L89-L108

```
function ERC1155RandomOut(uint256 amount, address to)
        external
        preventReentrant
        returns (uint256 paid)
    {
        (uint256 rawPay, ) = queryNFTRandomOut(amount);
        paid = IFilterAdmin(_OWNER_).burnFragFrom(msg.sender, rawPay);
        for (uint256 i = 0; i < amount; i++) {
            uint256 randomNum = _getRandomNum() % _TOTAL_NFT_AMOUNT_;
            uint256 sum;
            for (uint256 j = 0; j < _NFT_IDS_.length; j++) {
                sum += _NFT_RESERVE_[_NFT_IDS_[j]];
                if (sum >= randomNum) {
                    _transferOutERC1155(to, _NFT_IDS_[j], 1);
                    emit RandomOut( _NFT_IDS_[j], 1);
                    break;
                }
            }
        }
    }
```

解决方案

1. 建议随机数采用 chainlink 提供的随机数预言机来获取。
2. 建议获取随机的操作和确定 NFT 的编号操作分开进行，避免被回滚攻击。

漏洞状态

已确认

[N6] [高] 业务逻辑错误

漏洞类型: 业务逻辑缺陷审计

详细内容

FilterERC721V1 合约的 ERC721TargetOut 函数在执行 burnFragFrom 操作的时候传入的是 msg.sender , 但是该函数是由 DODONFTPoolProxy 合约的 erc721TargetOut 来进行调用的，存在逻辑上的错误，应该 burnFragFrom 的地址是用户的地址，如果让用户直接调用 FilterERC721V1 合约的 ERC721TargetOut 函数这样就无法对用户期望的 maxBurnAmount 进行管理。

- NFTPool/impl/FilterERC721V1.sol#L83-L112

```
function ERC721TargetOut (uint256[] memory tokenIds, address to)
        external
        preventReentrant
        returns (uint256 paid)
    {
        (uint256 rawPay, ) = queryNFTTargetOut(tokenIds.length);
        paid = IFilterAdmin(_OWNER_).burnFragFrom(msg.sender, rawPay);
        for (uint256 i = 0; i < tokenIds.length; i++) {
            _transferOutERC721(to, tokenIds[i]);

            emit TargetOut(tokenIds[i]);
        }
        _TOTAL_NFT_AMOUNT_ = _NFT_IDS_.length;
    }

function ERC721RandomOut(uint256 amount, address to)
        external
        preventReentrant
        returns (uint256 paid)
    {
        (uint256 rawPay, ) = queryNFTRandomOut(amount);
        paid = IFilterAdmin(_OWNER_).burnFragFrom(msg.sender, rawPay);
        for (uint256 i = 0; i < amount; i++) {
            uint256 index = _getRandomNum() % _TOTAL_NFT_AMOUNT_;
            _transferOutERC721(to, _NFT_IDS_[index]);

            emit RandomOut(_NFT_IDS_[index]);
        }
        _TOTAL_NFT_AMOUNT_ = _NFT_IDS_.length;
    }
```

- SmartRoute/proxies/DODONFTPoolProxy.sol#L85-L107

```
function erc721TargetOut(
        address filter,
        uint256[] memory tokenIds,
        address to,
        uint256 maxBurnAmount
    ) external {
        uint256 paid = IFilter(filter).ERC721TargetOut(tokenIds, to);
        require(paid <= maxBurnAmount, "BURN_AMOUNT_EXCEED");

        emit Erc721TargetOut(filter, to, paid);
    }

function erc721RandomOut(
        address filter,
        uint256 amount,
        address to,
        uint256 maxBurnAmount
    ) external {
        uint256 paid = IFilter(filter).ERC721RandomOut(amount, to);
        require(paid <= maxBurnAmount, "BURN_AMOUNT_EXCEED");

        emit Erc721RandomOut(filter, to, paid);
    }
```

同样的 FilterERC1155V1 合约也存在相同的问题。

- SmartRoute/proxies/DODONFTPoolProxy.sol#L128-L151

```
function erc1155TargetOut(
        address filter,
        uint256[] memory tokenIds,
        uint256[] memory amounts,
        address to,
        uint256 maxBurnAmount
    ) external {
        uint256 paid = IFilter(filter).ERC1155TargetOut(tokenIds, amounts, to);
        require(paid <= maxBurnAmount, "BURN_AMOUNT_EXCEED");

        emit Erc1155TargetOut(filter, to, paid);
    }

    function erc1155RandomOut(
        address filter,
```

```
        uint256 amount,
        address to,
        uint256 maxBurnAmount
    ) external {
        uint256 paid = IFilter(filter).ERC1155RandomOut(amount, to);
        require(paid <= maxBurnAmount, "BURN_AMOUNT_EXCEED");

        emit Erc1155RandomOut(filter, to, paid);
    }
```

- NFTPool/impl/FilterERC1155V1.sol#L74-L108

```
function ERC1155TargetOut(
        uint256[] memory tokenIds,
        uint256[] memory amounts,
        address to
    ) external preventReentrant returns (uint256 paid) {
        uint256 totalAmount = 0;
        for (uint256 i = 0; i < tokenIds.length; i++) {
            totalAmount += amounts[i];
            _transferOutERC1155(to, tokenIds[i], amounts[i]);
            emit TargetOut(tokenIds[i], amounts[i]);
        }
        (uint256 rawPay, ) = queryNFTTargetOut(totalAmount);
        paid = IFilterAdmin(_OWNER_).burnFragFrom(msg.sender, rawPay);
    }

    function ERC1155RandomOut(uint256 amount, address to)
        external
        preventReentrant
        returns (uint256 paid)
    {
        (uint256 rawPay, ) = queryNFTRandomOut(amount);
        paid = IFilterAdmin(_OWNER_).burnFragFrom(msg.sender, rawPay);
        for (uint256 i = 0; i < amount; i++) {
            uint256 randomNum = _getRandomNum() % _TOTAL_NFT_AMOUNT_;
            uint256 sum;
            for (uint256 j = 0; j < _NFT_IDS_.length; j++) {
                sum += _NFT_RESERVE_[_NFT_IDS_[j]];
                if (sum >= randomNum) {
                    _transferOutERC1155(to, _NFT_IDS_[j], 1);
                    emit RandomOut( _NFT_IDS_[j], 1);
                    break;
                }
```

```
            }
        }
    }
```

解决方案

1. 建议在 FilterERC721V1 和 FilterERC1155V1 合约中校验 TargetOut ， RandomOut 的调用者是 DODONFTPoolProxy 地址。
2. TargetOut ， RandomOut 接收 DODONFTPoolProxy 合约传入的 msg.sender 再传入 FilterERC721V1 和 FilterERC1155V1 合约中的 TargetOut ， RandomOut 函数时对地址进行检查，确保 burnFragFrom 的地址是经过检查的用户地址。

漏洞状态

已修复；经过沟通讨论，该问题已修改为仅允许用户从 FilterERC721V1 和 FilterERC1155V1 合约调用 TargetOut 和 RandomOut。

[N7] [建议] 同名函数的安全提醒

漏洞类型: 其它

详细内容

FilterAdmin 合约继承了 InitializableMintableERC20 合约，存在两个相同的 init 函数，可能会在后续的开发或代码交接过程中存在混淆的情况。

- external/ERC20/InitializableMintableERC20.sol#L29-L43

```
function init(
        address _creator,
        uint256 _initSupply,
        string memory _name,
        string memory _symbol,
        uint8 _decimals
    ) public {
        initOwner(_creator);
        name = _name;
        symbol = _symbol;
```

```
        decimals = _decimals;
        totalSupply = _initSupply;
        balances[_creator] = _initSupply;
        emit Transfer(address(0), _creator, _initSupply);
    }
```

- NFTPool/impl/FilterAdmin.sol#L32-L53

```
function init(
        address owner,
        uint256 initSupply,
        string memory name,
        string memory symbol,
        uint256 feeRate,
        address controller,
        address maintainer,
        address[] memory filters
    ) external {
        super.init(owner, initSupply, name, symbol, 18);
        _INIT_SUPPLY_ = initSupply;
        _FEE_RATE_ = feeRate;
        _CONTROLLER_ = controller;
        _MAINTAINER_ = maintainer;
        _FILTERS_ = filters;
        for (uint256 i = 0; i < filters.length; i++) {
            _FILTER_REGISTRY_[filters[i]] = true;
        }

        emit FilterAdminInit(owner, feeRate);
    }
```

解决方案

建议采用不同的函数命名避免在后续的开发迭代还有代码交接过程中出现理解上的偏差。

漏洞状态

已确认

[N8] [低] 算术运算没有使用 SafeMath

漏洞类型: 整数上溢/下溢攻击

详细内容

合约中有引用了 SafeMath 模块但是实际并没有使用 SafeMath 模块进行算术运算，且合约允许的编译版本是 pragma solidity 0.6.9;

解决方案

建议采用 SafeMath 模块进行算术运算或使用 pragma solidity 0.8.x; 以上的版本。

漏洞状态

已修复；该问题已经在 commit: 7b526b8208ec172b4ccfabbf22b39edcbe284561 中进行了修复。

[N9] [低] 函数可见性问题

漏洞类型: Gas优化设计

详细内容

queryMintFee 和 queryBurnFee 函数是用于读取数据的不会影响全局的变量，可见性应该为 view。

- NFTPool/impl/FilterAdmin.sol#L78-L105

```
function queryMintFee(uint256 rawAmount)
        public
        returns (
            uint256 poolFee,
            uint256 mtFee,
            uint256 afterChargedAmount
        )
    {
        uint256 mtFeeRate = IController(_CONTROLLER_).getMintFeeRate(address(this));
        poolFee = DecimalMath.mulFloor(rawAmount, _FEE_RATE_);
        mtFee = DecimalMath.mulFloor(rawAmount, mtFeeRate);
        afterChargedAmount = rawAmount.sub(poolFee).sub(mtFee);
    }

    function queryBurnFee(uint256 rawAmount)
        public
        returns (
            uint256 poolFee,
            uint256 mtFee,
```

```
            uint256 afterChargedAmount
        )
    {
        uint256 mtFeeRate = IController(_CONTROLLER_).getBurnFeeRate(address(this));
        poolFee = DecimalMath.mulFloor(rawAmount, _FEE_RATE_);
        mtFee = DecimalMath.mulFloor(rawAmount, mtFeeRate);
        afterChargedAmount = rawAmount.add(poolFee).add(mtFee);
    }
```

解决方案

建议在 queryMintFee 和 queryBurnFee 函数中添加 view 的可见性。

漏洞状态

已修复；该问题已经在 commit: 15675bf5f274ce00ed43e69eacde28400ae1f441 中进行了修复。

[N10] [严重] 开放式外部调用

漏洞类型: 不安全的外部调用

详细内容

代码中有添加了防止重入的锁，但是外部调用 `(bool success, ) = dodoProxy.call(dodoSwapData);` 没有遵循 Checks-Effects-Interactions 的规范。且 dodoProxy 外部调用的合约没有白名单限制，如果外部合约存在安全问题或外部合约是恶意的，可能会因为开放式外部调用带来新的风险。如果用户授权额度给 DODONFTApprove 没有及时使用或者没有及时清除，恶意攻击者可以利用外部调用转走 DODONFTApprove 多余的额度，参考 N4 的问题。

- SmartRoute/proxies/DODONFTPoolProxy.sol#L222-L251

```
function erc721ToErc20(
        address filterAdmin,
        address filter,
        address nftContract,
        uint256 tokenId,
        address toToken,
        address dodoProxy,
        bytes memory dodoSwapData
    )
        external
        preventReentrant
    {
```

```
        IDODONFTApprove(_DODO_NFT_APPROVE_).claimERC721(nftContract, msg.sender,
filter, tokenId);

        uint256[] memory tokenIds = new uint256[](1);
        tokenIds[0] = tokenId;

        uint256 receivedFragAmount = IFilter(filter).ERC721In(tokenIds,
address(this));

        _generalApproveMax(filterAdmin, _DODO_APPROVE_, receivedFragAmount);
        (bool success, ) = dodoProxy.call(dodoSwapData);
        require(success, "API_SWAP_FAILED");

        uint256 returnAmount = _generalBalanceOf(toToken, address(this));

        _generalTransfer(toToken, msg.sender, returnAmount);

        emit Erc721toErc20(nftContract, tokenId, toToken, returnAmount);
    }
```

解决方案

1. 建议遵循安全编码规范进行约束，遵循：先判断，后写入变量，再进行外部调用(Checks-Effects-Interactions)，将外部调用放在函数的最后。
2. 如果业务上需要将外部调用放在中间，建议为 dodoProxy 设置白名单的限制。
3. 不建议将授权额度设置为最大的额度，应该要根据实际需要使用的数量进行设置。

漏洞状态

已修复；该问题已经在 commit: 7b526b8208ec172b4ccfabbf22b39edcbe284561 中进行了修复。

5 审计结果

审计编号	审计团队	审计日期	审计结果
0X002109270001	SlowMist Security Team	2021.09.13 - 2021.09.20	高风险

总结：

慢雾安全团队采用人工结合内部工具对代码进行分析，审计期间发现了 1 个严重漏洞， 4 个高危漏洞， 1 个中危漏洞， 2 个低危漏洞， 2 个增强建议。其中 3 个高危漏洞， 1 个中危漏洞， 2 个增强建议已确认；其它所有漏洞均已修复。 目前代码还未部署到主网。

6 声明

厦门慢雾科技有限公司(下文简称 ”慢雾”) 仅就本报告出具前项目方已经发生或存在的事实出具本报告，并就此承担相应责任。对于出具以后项目方发生或存在的未知漏洞及安全事件，慢雾无法判断其安全状况，亦不对此承担责任。本报告所作的安全审计分析及其他内容，仅基于信息提供者截至本报告出具时向慢雾提供的文件和资料(简称“已提供资料”)。慢雾假设: 已提供资料不存在缺失、被篡改、删减或隐瞒的情形。如已提供资料信息缺失、被篡改、删减、隐瞒或反映的情况与实际情况不符的，慢雾对由此而导致的损失和不利影响不承担任何责任，慢雾仅对该项目的安全情况进行约定内的安全审计并出具了本报告，慢雾不对该项目背景及其他情况进行负责。

官方网址

www.slowmist.com

电子邮箱

team@slowmist.com

微信公众号