Making Web3 Space Safer for Everyone

KALOS

Agent Exchange & Pool

Security Assessment

Published on : 22 May. 2024
Version v1.0

Security Report Published by KALOS

v1.0 22 May. 2024

Auditor: Hun

Found issues

Severity of Issues	Findings	Resolved	Acknowledged	Comment
Critical	2	2	-	-
High	2	2	-	-
Medium	1	1	-	-
Low	1	1	-	-
Tips	7	7	-	-

TABLE OF CONTENTS

ABOUT US

Making Web3 Space Safer for Everyone

KALOS 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.

Having secured $60B crypto assets on over 400 main-nets, Defi protocols, NFT services, P2E, and Bridges, KALOS 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@kalos.xyz
Website: https://kalos.xyz

Executive Summary

Purpose of this report

This report was prepared to audit the security of the Agent Exchange and Pool contracts. KALOS conducted the audit focusing on whether the system created by the Agent Exchange is soundly implemented and designed as specified in the materials, in addition to the safety and security of the Agent Exchange and Pool.
In detail, we have focused on the following

- Known smart contract vulnerabilities
- Unintended behavior in the trade process
- Asset theft from the contract
- Risks in the signature verification process
- Risks in the message signing process on the user side

Codebase Submitted for the Audit

The codes used in this Audit can be found on GitHub (https://github.com/AgentExchange/agent-contracts/tree/e23a0970269fc5386bc83868c0d693d8aa902708).

The last commit of the code used for this Audit is “eff6f89df74ff6b20253fe2d7048ced1d86a7f52”.

Audit Timeline

Date	Event
2024/5/16	Audit Initiation (Exchange and Pool)
2024/5/22	Delivery of v1.0 report.

Findings

KALOS found 2 Critical, 2 High, 1 Medium, and 1 Low severity issues. There are 7 Tips issues explained that would improve the code's usability or efficiency upon modification.

Severity	Issue	Status
Critical	1. Signature Replay due to insufficient message structure leads to Loss of User Assets	(Fixed - v1.0)
High	2. An Opaque Signature Message Without EIP712 Typed Struct Data Allows Users to Sign Malicious Messages	(Fixed - v1.0)
Low	3. Potential Bid Collision Due to Lack of Unique ID in Bid Structure can result in Trading Failure.	(Fixed - v1.0)
Critical	4. Arbitrary signature message leads to Loss of User Assets	(Fixed - v1.0)
Medium	5. Wrong listing delete in the takeAsk() disables users from relisting their item	(Fixed - v1.0)
High	6. Potential Race Condition between front-end and on-chain can lead users to buy items at a higher price	(Fixed - v1.0)
Tips	7. Tips	(Fixed - v1.0)

Remarks

Only contracts are in scope. Off-chain such as the signing process on a back-end and marketplace front-end is not out of scope for this audit.

There is a centralization risk that can be referred to in the Access Controls section.

OVERVIEW

Protocol overview

• **AgentPool**

AgentPool is a smart contract that manages user balances for ERC20 tokens and Ether. It has deposit, withdrawal, and transfer functionality. It manages user balances for each token and has access control to ensure that only the AgentExchangeV1 contract can use the transfer function. The contract also provides a fund recovery feature that transfers a user's funds to the treasury. This function is restricted to the contract owner.

• **AgentExchangeV1**

AgentExchangeV1 is a trading platform for NFTs. It includes functionalities for listing NFTs, bidding, accepting asks, and accepting bids through signature verification. Users can list their NFTs for sale, and the contract ensures that only NFTs from approved collections can be listed. The NFT is mainly the AgentAccountToken. Bids can be placed on these NFTs and bidders have to have sufficient balances in the AgentPool. The takeAsk() function allows buyers to accept the listed price of an NFT. The takeBid() function enables NFT sellers to accept bids. Signature verification authenticates the bids and the asks. Bidder signatures ensure that bids are authorized by the bidder, and oracle signatures validate trading fee data including the fee discount rate provided by an external oracle.

Scope

```
├── contracts
│   ├── exchangeV1
│   │   ├── AgentExchangeV1.sol
│   │   └── AgentPool.sol
```

Access Controls

Agent contracts have the following access control mechanisms.

- AgentExchangeV1.onlyOwner()
- AgentPool.onlyOwner()
- AgentPool.onlyExchange()

onlyOwner(): The owner can approve collections and set crucial values such as the fee rate, fee oracle address, and exchange address. Also, they can also use the rescue fund function. The owner can transfer any tokens to the treasury by using this function. The owner and the treasury would be a multisig contract.

onlyExchange(): Only the exchange can transfer tokens in the agent pool between users.

The owner has permissions that can change the crucial part of the system and can transfer any tokens to the treasury. It is highly recommended to maintain the private key as securely as possible and strictly monitor the system state changes.

FINDINGS

1. Signature Replay due to insufficient message structure leads to Loss of User Assets

ID: EXCHANGE-01	Severity: Critical
Type: N/A	Difficulty: Low

Impact

An attacker can buy the relisted NFT at a lower price than a lister intended by replaying the Oracle signature with a message including the lower price.

Description

Exchange verifies two types of signatures, such as bidder and fee oracle signatures.

- The bidder signature is used to verify a bidder's bid and a lister's listed item when a lister accepts the bid.
- The fee oracle signature is used to verify the listed item and fee oracle's fee data which includes a fee discount rate when user accepts a bid or ask.

There are several problems found in verifying these signatures.

```
function _verifyOracleSignature(Listing memory item, Fees memory fee, bytes memory oracleSignature)
    internal
    view
    returns (bool)
{
    if (fee.expiry < block.timestamp) {
        return false;
    }
    bytes32 hash = keccak256(abi.encode(item, fee));
    return ECDSA.recover(hash, oracleSignature) == feeOracle;
}

function _hash(Bid memory bid, Listing memory item) internal pure returns (bytes32) {
    return keccak256(abi.encode(bid, item));
}
```

[https://github.com/AgentExchange/agent-contracts/blob/e23a0970269fc5386bc83868c0d693d8aa902708/contracts/exchangeV1/AgentExchangeV1.sol#L236-L250]

Both signatures do not include a unique ID such as nonce and salt, caller address, and domain information such as contract address and chain ID.

This causes several problems.

1. The Oracle signature can be reused in both takeBid() and takeAsk() functions.
2. The Oracle signature can be used by any user.
3. The Oracle signature can be reused on other chains and in other contracts.

These problems lead to loss of user's assets.

Attack Scenario

1. A seller sells NFT and re-lists NFT at a higher price than the sold price.
2. Seller buys back the sold NFT.
3. An attacker can buy the relisted NFT at a lower price by replaying the Oracle signature with a message including the lower price.

PoC code

```
function testTakeAskAttack() public {
    testTemps memory t = _testTemps();
    assert(agentTokenV1.ownerOf(t.tokenId) == address(t.owner));
    IAgentExchangeV1Utils.Listing memory l = _listItem(t);
    (IAgentExchangeV1Utils.Fees memory _fees, bytes memory signature) = _getFeesTemp(l);
    _addBalance(t.token, address(this), l.amount);

    assert(pool.balanceOf(l.token, treasury) == 0);
    assert(agentTokenV1.ownerOf(t.tokenId) == address(exchange));
    uint256 fees = l.amount * fees * (100 - _fees.discount) / 1000000;
    vm.expectEmit(true, true, true, true);
    emit AskTaken(address(this), l.seller, l.amount, l.token, l.tokenId, fees);
    exchange.takeAsk(l, _fees, signature);
    assert(pool.balanceOf(l.token, treasury) == fees);
    assert(pool.balanceOf(l.token, l.seller) == l.amount - fees);
    assert(agentTokenV1.ownerOf(t.tokenId) == address(this));
    // based on testTakeAsk()

    address currentOwner = agentTokenV1.ownerOf(t.tokenId);
    agentTokenV1.setWhitelist(currentOwner, true); // Suppose seller buys back.
    vm.prank(currentOwner);
    agentTokenV1.transferFrom(currentOwner, t.owner, t.tokenId); // seller bought back.
    t.price *= 10;
    _listItem(t); // Seller listed again at higher price within 1 day (fee.expiry)
    _addBalance(t.token, address(this), l.amount);
     // attacker can buy token for lower price re-using signature used to take bid or ask
    exchange.takeAsk(l, _fees, signature);
}
```

[PoC]

Recommendation

Use nonce or salt to prevent signature replay and add the caller address to the message to authenticate the taker.

Fix Comment

Fixed to add salt to the message.

2. An Opaque Signature Message Without EIP712 Typed Struct Data Allows Users to Sign Malicious Messages

ID: EXCHANGE-02	Severity: High
Type: Visibility	Difficulty: Low

Impact

The user sees only bytes of the message and may sign a malicious message without noticing it.

Description

```
    bytes32 hash = keccak256(abi.encode(item, fee));
    return ECDSA.recover(hash, oracleSignature) == feeOracle;
}

function _hash(Bid memory bid, Listing memory item) internal pure returns (bytes32) {
    return keccak256(abi.encode(bid, item));
}
```

[https://github.com/AgentExchange/agent-contracts/blob/e23a0970269fc5386bc83868c0d693d8aa902708/contracts/exchangeV1/AgentExchangeV1.sol#L244-L249]

The bidder signature does not use typed struct data of EIP712. The message consists only of concatenated bytes.

This message will not be clearly displayed to a user when the user signs it.

Recommendation

Use EIP712 typed structured data

Fix Comment

Fixed to use EIP712 typed structured data.

3. Potential Bid Collision Due to Lack of Unique ID in Bid Structure can result in Trading Failure.

ID: EXCHANGE-03
Severity: Low
Type: N/A
Difficulty: Low

Impact

This issue could prevent valid bids from being accepted and cause potential loss of trading opportunities.

Description

```
struct Bid {
    address bidder;
    address token;
    uint256 amount;
    uint256 expiry;
}
```

[https://github.com/AgentExchange/agent-contracts/blob/e23a0970269fc5386bc83868c0d693d8aa902708/contracts/interfaces/IAgentExchangeV1Utils.sol#L13-L18]

The bid signature verification function does not allow taking a bid if a duplicate bid with the same bid and item information already exists. Since the Bid structure does not include a unique ID, if a bidder cancels a bid and submits a new bid with the same bidder, token, amount, and expiry for the same item, then the bid cannot be accepted.

Recommendation

Add salt to the Bid message.

Fix Comment

Fixed to add salt to the Bid message.

4. Arbitrary signature message leads to Loss of User Assets

ID: EXCHANGE-04	Severity: Critical
Type: N/A	Difficulty: Low

Impact

This issue allows attackers to exploit outdated signatures to purchase NFTs at lower prices, causing losses for listers.

Description

```
function takeAsk(Listing calldata item, Fees calldata _fee, bytes calldata oracleSignature) external {
    if (
        listings[item.nftAddress][item.tokenId].seller == address(0)
            || msg.sender == listings[item.nftAddress][item.tokenId].seller
            || !_verifyOracleSignature(item, _fee, oracleSignature)
```

[https://github.com/AgentExchange/agent-contracts/blob/e23a0970269fc5386bc83868c0d693d8aa902708/contracts/exchangeV1/AgentExchangeV1.sol#L121-L125]

Attack Scenario

1. A lister lists their NFT at 1 ETH.
2. An attacker attempts to take this ask and gets the signature at the 1 ETH price.
3. The lister cancels and relists the NFT at a higher price of 2 ETH.
4. The attacker can still buy it for 1 ETH using the previously obtained signature, since _verifyOracleSignature() verifies the item parameter instead of the lister's actual listing.

Recommendations

1. The _verifySignature() and _verifyOracleSignature() functions must verify the listings stored in storage instead of the item parameter passed to them.
2. the takeBid and takeAsk functions should use item details loaded from storage.

Fix Comment

Fixed to use the item loaded from the storage.

5. Wrong listing delete in the takeAsk() disables users from relisting their item

ID: EXCHANGE-05
Severity: Medium
Type: N/A
Difficulty: Low

Impact

A canceled NFT cannot be re-listed permanently.

Description

```
delete (listings[item.token][item.tokenId]);
```

[https://github.com/AgentExchange/agent-contracts/blob/e23a0970269fc5386bc83868c0d693d8aa902708/contracts/exchangeV1/AgentExchangeV1.sol#L135]

The takeAsk() and takeBid() functions delete the listing with the ERC20 token address (item.token), not the NFT address (nftAddress).

This causes the already listed error when users relist their NFT because their listing was not deleted.

Recommendation

Use nftAddress instead of item.token to delete listing.

Fix Comment

Fixed to use nftAddress.

6. Potential Race Condition between front-end and on-chain can lead users to buy items at a higher price

ID: EXCHANGE-06
Severity: High
Type: N/A
Difficulty: Low

Impact

This issue may cause users to pay more than expected for an NFT, resulting in a loss.

Description

The takeAsk() function can cause inconsistency like race condition between the front-end and on-chain since it does not check whether the item parameter matches the actual listing.

Attack Scenario
1. An attacker lists their NFT at a low price.
2. A victim who already deposited various tokens navigates to the NFT's detail page on the front-end.
3. The attacker updates the listing with a more expensive token and a higher price.
4. The victim requests a fee oracle signature. The fee oracle fetches the updated (higher priced) listing using only the nftAddress and tokenId in parameters .
5. The fee oracle provides the signature to the victim
6. The victim ends up sending a transaction to take the ask at a higher price than expected because takeAsk does not check if the item parameter is the same as the actual listing.

Recommendation

The takeAsk() function should check that the token and amount parameters still match the listing's values like Uniswap's mitigations against sandwich attack.

Fix Comment

Fixed to add checks.

7. Tips

ID: EXCHANGE-07
Severity: Tips
Type: N/A
Difficulty: N/A

This section describes some tips to improve functionality and security in the future.

1. If the pool receives a deflationary token like a fee-on-transfer token, then the pool may become insolvent.

Recommendation
- The agent pool should use balance difference when receiving ERC20 tokens.

Fix Comment
- The pool will not support deflationary tokens.

2. If an owner sets the fee rate higher than the maximum fee rate, all trades may fail.

Recommendation
- Limit the fee rate below the maximum fee rate.

Fix Comment
- Fixed

3. If an ERC20 precompile is deployed at zero address on any chain, the deposited token cannot be withdrawn.

Recommendation
- The deposit() function should check if the token address is zero address.
- Use EIP-7528 for native asset address.

Fix Comment
- Fixed
- EIP-7528 is still not used.

4. The exchange should be able to change the fee oracle address in case the fee oracle account is compromised.

Recommendation

- Add setFeeOracle function

Fix Comment
- Fixed

5. Listing can also support the expiry to enable users to set a deadline for their listings.

Recommendation
- Add the expiry to the Listing struct

Fix Comment
- Fixed

6. Precision of the fee rate is too low due to the loss of the treasury's fee If the amount with too low decimals and the discount

Recommendation
- Increase the precision instead of 4 decimals.

Fix Commend
- Fixed

7. Event emission is missing in cancelBid()

Recommendation
- Add event emission

Fix Comment
- Fixed

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 codes, 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