AI Agent Layer Security Review

Duration: November 9, 2024 - November 18, 2024

Date: February 4, 2025

Conducted by: **KeySecurity**
gkrastenov, Lead Security Researcher

Table of Contents

1 About KeySecurity

KeySecurity is a new, innovative Web3 security company that hires top-talented security researchers for your project. We have conducted over 30 security reviews for various projects, collectively holding over $300,000,000 in TVL. For security audit inquiries, you can reach out to us on Twitter/X or Telegram `@gkrastenov` or check our previous work `here`.

2 About AI Agent Layer

`AI Agent Layer` supports a dynamic ecosystem of autonomous AI agents. On the platform, you can create AI agents by leveraging data from X and user-provided information. Each AI Agent is tokenized and integrated with the ecosystem's native token ($AIFUN).

3 Disclaimer

Audits are a time, resource, and expertise bound effort where trained experts evaluate smart contracts using a combination of automated and manual techniques to identify as many vulnerabilities as possible. Audits can show the presence of vulnerabilities **but not their absence**.

4 Risk classification

Severity	Impact: High	Impact: Medium	Impact: Low
Likelihood: High	Critical	High	Medium
Likelihood: Medium	High	Medium	Low
Likelihood: Low	Medium	Low	Low

4.1 Impact

- **High** - leads to a significant loss of assets in the protocol or significantly harms a group of users.
- **Medium** - only a small amount of funds can be lost or a functionality of the protocol is affected.
- **Low** - any kind of unexpected behaviour that's not so critical.

4.2 Likelihood

- **High** - direct attack vector; the cost is relatively low to the amount of funds that can be lost.
- **Medium** - only conditionally incentivized attack vector, but still relatively likely.
- **Low** - too many or too unlikely assumptions; provides little or no incentive.

4.3 Actions required by severity level

- **Critical** - client **must** fix the issue.
- **High** - client **must** fix the issue.
- **Medium** - client **should** fix the issue.
- **Low** - client **could** fix the issue.

5 Executive summary

Overview

Project Name	AI Agents Fun
Repository	https://github.com/AIAgentsFun/evm-smart-contracts
Commit hash	74895260479d21aeab195dc725700285a3dd7b63
Review Commit hash	98512594cc144ba534c161da33175f029e141fb3
Documentation	N/A
Methods	Manual review

Scope

AIFunAccessControl.sol
Agent.sol
Badge.sol
BondingCurve.sol
CollectionsRegistry.sol
Factory.sol
FundSplitter.sol
Router.sol
SystemContext.sol
Treasury.sol

Timeline

November 9, 2024	Audit kick-off
November 18, 2024	Preliminary report
November 18, 2024	Mitigation review

Issues Found

Severity	**Count**
High	5
Medium	3
Low	2
Information	7
Total	**17**

6 Findings

6.1 High

6.1.1 The wrong denominator is used for calculating the owner's claim amount

Severity: *High*

Context: FundSplitter.sol#L185-L186

Description: When the owner decides to call the `claimAllocation` function, he should receive 5% of the total supply as a fee. The wrong denominator is used to calculate his fee. Instead of using the correct `DENOMINATOR`, `totalSupply` is used again.

```
uint256 ownerAmount = (pack.agent.totalSupply() * OWNER_FEE_NOMINATOR) /
            pack.agent.totalSupply();
```

This will always return `500`, and the `Treasury` contract will transfer only 500 agents to the owner instead of 5% of the total supply.

Recommendation: Use `DENOMINATOR` instead of `pack.agent.totalSupply();`.

Resolution and Client comment: Resolved. Commit: #c261d8fbd8d863ae319dfbc00a82223233d8030f

6.1.2 Liquidity provision can be blocked

Severity: *High*

Context: Router.sol#L325

Description: In the Router contract, liquidity provision can be blocked. For example, if someone creates a token and, within the next 6 hours, 980 shares are bought, and I then decide to buy the last 20 shares, the validation function `_validateDexLaunchStatus` should check the status. If the status is **INVALID** or **FAILED**, the whole transaction will revert.

The following things are made in `dexLaunchStatusAfter`:

- The sum of `totalSharesAfter` will be `badge.totalDistributedShares()+ shares`, which is equal to `980 + 20 = 1,000`.
- The return status from the `_dexLaunchStatus` function will be **READY_TO_ADD_LIQUIDITY**, meaning the next step after buying the tokens will be liquidity provision.

After the purchase, the corresponding 20 shares will be added, and `badge.totalDistributedShares()` will now be `1,000`.

```
        int256 sharesToMint = _calculateSharesToMint(badge, buyer, shares);

        _processBuyTransaction(pack, buyer, guid, params);
        badge.mint(buyer, sharesToMint);

        if (_isReadyToAddLiquidity(badge, shares)) {
            require(amountOut > 0, InvalidAmountOut());
            _processLiquidityProvision(guid, amountOut, pack, badge);
        }
```

When the `_isReadyToAddLiquidity` function is called, the `dexLaunchStatusAfter` function will return an **INVALID** status.

This is because, at that time, `badge.totalDistributedShares()` will be `1,000`, and adding 20 more shares will result in a sum of `1,020`. Since the condition `shares > badge.maxShares()` will evaluate to **true**, the status will return **INVALID**, which will block liquidity provision.

```
function _dexLaunchStatus(
        IBadge badge,
        uint256 deadline, // now + 1 days
        uint256 shares    // 1_20
    ) internal view returns (DexLaunchStatus) {
        if (shares > badge.maxShares()) {
        //@audit-info possible problem
            return DexLaunchStatus.INVALID;
        } else if (shares == badge.maxShares()) {
            return
                liquidityProvidedFor(
                    _collectionsRegistry().getGuidByBadge(badge)
                )
                    ? DexLaunchStatus.COMPLETED
                    : DexLaunchStatus.READY_TO_ADD_LIQUIDITY;
        } else if (block.timestamp > deadline) {
            return DexLaunchStatus.FAILED;
        }
```

Recommendation: Calculate correctly `totalDistributedShares`.

Resolution and Client comment: Resolved. PR: #6

6.1.3 The owner of the collection can not claim their allocation

Severity: *High*

Context: FundSplitter.sol#L188-L192

Description: The owner of the collection can not claim their allocation because, in the claimAllocation function, the agent's tokens are attempted to be transferred from the `Treasury` to the owner. However, this is not possible. Before providing liquidity, the agent's tokens are pulled from the `Treasury` and transferred to the FundSplitter contract. Therefore, the agent's token allocation is in the `FundSplitter` contract, not in the Treasury.

Recommendation: Transfer tokens from the `FundSplitter` contract, not from the `Treasury` contract.

Resolution and Client comment: Resolved. Commit: #768b0d2ce8c68df79467e5ac9f309955467f46d6

6.1.4 User can not sell their tokens when DEX launch deadline has expired

Severity: *High*

Context: Router.sol#L116

Description: The user can not sell back their shares to the bonding curve when the DEX launch deadline has expired. The status will be marked as `FAILED`, and their sell transaction will revert. This will result in the loss of 100% of their funds instead of paying only a 1% tax on the sale and receiving the remaining funds back.

Recommendation: Allow the user to sell tokens when the DEX launch deadline has expired.

Resolution and Client comment: Resolved. PR: #11

6.1.5 The wrong pool is used for checking liquidity

Severity: *High*

Context: Router.sol#L258

Description: Before providing liquidity, it is checked if the pool already has liquidity. If it has, the provision of liquidity will not be made. Currently, the wrong pool is used for checking liquidity. Instead of checking the liquidity in the Agent <-> Abbys pool, it is checked for WETH <-> Agent.

Recommendation: Use the address of the Abbys token instead of WETH.

Resolution and Client comment: Resolved. PR: #11

6.2 Medium

6.2.1 The owner of the collection is not stored

Severity: *Medium*

Context: CollectionsRegistry.sol#L34-L47

Description: When a new collection is registered through the `registerCollection` function, it is provided with the address of the owner. Currently, the provided owner address is not stored in the `_owners` mapping.

This will also cause the `getOwner()` function in the `CollectionsRegistry` contract to always return `address(0)`.

Recommendation: Store the owner of the new collection in the `_owners` mapping.

Resolution and Client comment: Resolved. PR: #1

6.2.2 The collection owner can block the FundSplitter logic

Severity: *Medium*

Context: FundSplitter.sol#L80

Description: The collection owner can block the FundSplitter logic when the agentCreatorFee is sent to them. If the owner is a malicious smart contract that created the collection, they can stop ETH from being received. If a user sells all their purchased shares, they must pay a 10% fee on the refunded amount.

Recommendation: Does not handle call to collection owner.

Resolution and Client comment: Resolved. Commit: #768b0d2ce8c68df79467e5ac9f309955467f46d61

6.2.3 The view function dexLaunchStatus uses the wrong value for shares

Severity: *Medium*

Context: Router.sol#L291

Description: The view function `dexLaunchStatus` calls the internal function `_dexLaunchStatus` to return the current status by providing the badge, DEX deadline, and badge supply.

The `badge.totalSupply()` function returns the number of minted NFTs, not the number of shares already purchased. In `_dexLaunchStatus`, it checks if `shares >= badge.maxShares()` rather than the count of minted NFTs.

```
  _dexLaunchStatus(
                badge,
                dexLaunchDeadlineAt(guid),
                badge.totalSupply()
            );


function _dexLaunchStatus(
       IBadge badge,
       uint256 deadline, // now + 1 days
       uint256 shares    // 10
   ) internal view returns (DexLaunchStatus) {
       if (shares > badge.maxShares()) {
           return DexLaunchStatus.INVALID;
       } else if (shares == badge.maxShares()) {
```

Recommendation: Use `badge.totalDistribiutedShares()` instead of `badge.totalSupply()`.

Resolution and Client comment: Resolved. PR: #12

6.3 Low

6.3.1 Missing check for tokenId existence

Severity: *Low*

Context: Badge.sol#L110

Description: When the `tokenURI` function is called in the Badge contract, it never checks whether the given tokenId exists. According to the recommendation in EIP-721, the `tokenURI` function should throw an error if the tokenId is not a valid NFT. This recommendation is followed in the OZ ERC721 Implementation contract, as well as the ERC721A implementation.

```
    function tokenURI(
        uint256 tokenId //@audit-issue missing check for existence
    ) public view override(IERC721A, ERC721A) returns (string memory) {
        return string.concat(_baseTokenURI, tokenId.toString());
    }
```

Recommendation: Check if the given `tokenId` exists, if it is invalid, revert with a custom error.

Resolution and Client comment: Resolved. PR: #7

6.3.2 sellParams can return wrong value

Severity: *Low*

Context: BondingCurve.sol#L88-L106

Description: The user should not be able to sell more shares than they have already bought. Currently, the `sellParams` function returns an incorrect value for the total refund when `shares > alreadySoldShares`.

Example: → jump = $1; firstPriceToken = $6, alreadySoldShares = 1, shares = 3 result: 15; it should be 6 + 7 = 13

→ jump = $1; firstPriceToken = $6, alreadySoldShares = 5, shares = 10 result: 55,;it should be 40

Recommendation: Verify that the number of `shares` is always <= the `alreadySoldShares`.

Resolution and Client comment: Resolved. PR: #8

6.4 Information

6.4.1 Emit event in crucial place

Severity: *Information*

Context: Agent.sol#L73

Description: Emit an event in crucial places, such as in the makeTransferable() function, when the transferable functionality of the Agent is set to true.

```
    function makeTransferable() external override onlyTransferControllerAccess {
        //@audit-issue emit event
```

Recommendation: Emit event in makeTransferable() function.

Resolution and Client comment: Resolved. PR: #9

6.4.2 Use _totalSupply directly instead of calling totalSupply() internally

Severity: *Information*

Context: Agent.sol#L61

Description: In the constructor of the Agent contract, the total supply of the token is provided directly. Use the _totalSupply variable instead of internally calling the totalSupply() function when the initial supply is minted.

```
_mint(_mintTo, totalSupply()); //@audit-issue use directly _totalSupply
```

Recommendation: Use `_totalSupply` directly instead of calling `totalSupply()` internally.

Resolution and Client comment: Resolved. PR: #10

6.4.3 The onlySystemContextAdmin modifier is redundant in the _setContract() function

Severity: *Information*

Context: SystemContext.sol#L83

Description: The `onlySystemContextAdmin` modifier is redundant in the internal `_setContract()` function. The `_setContract()` function is called only from the `setContract()` and `setContractByName()` functions, both of which already have the `onlySystemContextAdmin` modifier.

Recommendation: Allow the admin to update sticker prices.

Resolution and Client comment: Resolved. PR: #13

6.4.4 Missing collectionInitialized modifier for sellParams function

Severity: *Information*

Context: BondingCurve.sol#L88

Description: The bonding curve should work only with an initialized collection. The `collectionInitialized` modifier is used to check if the collection is initialized. This modifier is applied to the `buyParams` function but not to the `sellParams` function. Consequently, the function will always return 0 for `totalPrice` and `agentsToTransfer`.

Recommendation: Add `collectionInitialized` modifier to the `sellParams` function.

Resolution and Client comment: Resolved. PR: #14

6.4.5 Redundant errors

Severity: *Information*

Context: BondingCurve.sol#L9-L10

Description: In the `AiAgentsLayerBondingCurve` contract, errors `TotalToCostAlreadySet()` and `ZeroTotalToCollect()` are never used.

Recommendation: Remove the redundant error.

Resolution and Client comment: Acknowledged.

6.4.6 Some internal functions in the FundSplitter contract are never used

Severity: *Information*

Context: FundSplitter.sol#L201-L205

Description: The _getWETHAmountLiquidity and _sendFunds functions in the FundSplitter contract are never used.

```
        // amount we swap for ABYSS token (92.85%) = 9_285
        uint256 swapAmount = (collected.native * LIQUIDITY_NOMINATOR) /
            DENOMINATOR;


 function _getWETHAmountLiqudity(
        uint256 amount
    ) internal pure returns (uint256) {
        return (amount * LIQUIDITY_NOMINATOR) / DENOMINATOR;
    }


function _sendFunds(ERC20 agent, uint256 amount, address to) internal {
        SafeTransferLib.safeTransfer(agent, to, amount);
    }
```

Recommendation: Remove redundant functions.

Resolution and Client comment: Resolved. PR: #15

6.4.7 AI_AGENT_FUN_PACK_CREATOR_ROLE role is granted twice for the router

Severity: *Information*

Context: DeployAIFun.s.sol#L93-L107

Description: The `AI_AGENT_FUN_PACK_CREATOR_ROLE` role is granted twice for the router.

Recommendation: Grant the `AI_AGENT_FUN_PACK_CREATOR_ROLE` role to the router only once.

Resolution and Client comment: Resolved. PR: #16