OpenZeppelin | security

Capx AI EVM Contracts Audit

October 24, 2025

Table of Contents

Summary

Type	Token
Timeline	From 2025-10-15 To 2025-10-22
Languages	Solidity

Total Issues	7 (7 resolved)
Critical Severity Issues	0 (0 resolved)
High Severity Issues	0 (0 resolved)
Medium Severity Issues	0 (0 resolved)
Low Severity Issues	1 (1 resolved)
Notes & Additional Information	6 (6 resolved)

Scope

OpenZeppelin performed an audit of the Capx-AI/Capx-AI-EVM-Contracts repository at commit b2e085e.

In scope were the following files:

```
contracts
├── CapxAgentClone.sol
├── CapxAgentFactory.sol
└── abstract
    └── CapxOwnableClone.sol
```

System Overview

Capx is an Ethereum Layer 2 network, purpose-built to create, deploy, and monetize AI apps. The contracts in scope allow the Capx Tech team to create tradable ERC-20 tokens for AI agents within the Capx AI Ecosystem.

The `CapxAgentFactory` contract is a factory contract that allows for the efficient creation of new Agents proxies. The `CapxAgentClone` contract is the implementation contract that the proxies created by the factory will point to. Each Agent proxy is represented by its own unique, standard ERC-20 token. The factory uses the minimal proxy pattern to deploy these agent contracts gas-efficiently.

This means that instead of deploying a full copy of the contract logic each time, it deploys a minimal proxy that delegates all its logic to an implementation of `CapxAgentClone`. This architecture allows any developer or user to create a token for their AI application, enabling novel monetization strategies, community governance, and access control for their AI services.

Security Model and Trust Assumptions

Each `CapxAgentClone` contract has an associated owner which is specified upon initialization by the `CapxAgentFactory` contract, though no access-controlled functionality is currently present that is exclusive to the owner of the contract. Instead of assigning the owner minting capabilities, the token's total supply is predefined and minted to the owner upon initialization. The `CapxAgentFactory` contract which is responsible for creation of the various `CapxAgentClone` contracts is both ownable and upgradeable, using OpenZeppelin's open-source library of contracts. Only the owner of the `CapxAgentFactory` contract can create new `CapxAgentClone` contracts.

Privileged Roles

Throughout the codebase, the following privileged roles were identified:

- `CapxAgentClone` owner: No current permissioned capabilities.
- `CapxAgentFactory` owner: It can upgrade the factory contract and create new `CapxAgentClone` contracts.

Low Severity

L-01 The `owner` Role of `CapxAgentClone` Is Unused

The `CapxAgentClone` contract inherits the `Ownable` contract and assigns the role to an address in the `initialize` function. However, it does not use the role for any functionality.

Consider not inheriting the `Ownable` contract in `CapxAgentClone` to improve code clarity and maintainability.

Update: *Resolved in pull request #23 at commit 30522c6.*

Notes & Additional Information

N-01 Use `calldata` Instead of `memory`

When dealing with the parameters of `external` functions, it is more gas-efficient to read their arguments directly from `calldata` instead of storing them to `memory`. `calldata` is a read-only region of memory that contains the arguments of incoming `external` function calls. This makes using `calldata` as the data location for such parameters cheaper and more efficient compared to `memory`. Thus, using `calldata` in such situations will generally save gas and improve the performance of a smart contract.

Within `CapxAgentFactory.sol`, the `_agentParams` parameter should use `calldata` instead of `memory`.

Consider using `calldata` as the data location for the parameters of `external` functions to optimize gas usage.

Update: *Resolved in pull request #24 at commit 720f903.*

N-02 Missing Security Contact

Providing a specific security contact (such as an email address or ENS name) within a smart contract significantly simplifies the process for individuals to communicate if they identify a vulnerability in the code. This practice is quite beneficial as it permits the code owners to dictate the communication channel for vulnerability disclosure, eliminating the risk of miscommunication or failure to report due to a lack of knowledge on how to do so. In addition, if the contract incorporates third-party libraries and a bug surfaces in those, it becomes easier for their maintainers to contact the appropriate person about the problem and provide mitigation instructions.

Throughout the codebase, multiple instances of contracts not having a security contact were identified:

- The `CapxAgentClone` contract
- The `Agent` abstract contract
- The `CapxAgentFactory` contract
- The `Ownable` abstract contract

Consider adding a NatSpec comment containing a security contact above each contract definition. Using the `@custom:security-contact` convention is recommended as it has been adopted by the OpenZeppelin Wizard and the ethereum-lists.

Update: *Resolved in pull request #25 at commit 902fc08.*

N-03 Use Custom Errors

Since Solidity version 0.8.4, custom errors provide a cleaner and more cost-efficient way to explain to users why an operation failed.

Throughout the codebase, multiple instances of `revert` and/or `require` messages were identified:

- In the `CapxAgentClone.sol` file, the `require(owner_ != address(0), "InvalidOwnerAddress")` statement
- In the `CapxAgentClone.sol` file, the `require(decimals_ >= 6, "DecimalsTooLow")` statement
- In the `CapxAgentClone.sol` file, the `require(totalSupply_ > 0, "ZeroSupply")` statement

- In the `CapxAgentClone.sol` file, the `require( currentAllowance >= subtractedValue, "ERC20: decreased allowance below zero" )` statement
- In the `CapxAgentClone.sol` file, the `require(sender != address(0), "ERC20: transfer from the zero address")` statement
- In the `CapxAgentClone.sol` file, the `require(recipient != address(0), "ERC20: transfer to the zero address")` statement
- In the `CapxAgentClone.sol` file, the `require( senderBalance >= amount, "ERC20: transfer amount exceeds balance" )` statement
- In the `CapxAgentClone.sol` file, the `require(owner != address(0), "ERC20: approve from the zero address")` statement
- In the `CapxAgentClone.sol` file, the `require(spender != address(0), "ERC20: approve to the zero address")` statement
- In the `CapxAgentClone.sol` file, the `require( currentAllowance >= amount, "ERC20: transfer amount exceeds allowance" )` statement
- In the `CapxAgentFactory.sol` file, the `require(_newImplementation != address(0), "ZeroAddress")` statement
- In the `CapxAgentFactory.sol` file, the `require(_newImplementation.code.length > 0, "NotContract")` statement
- In the `CapxAgentFactory.sol` file, the `require(_owner != address(0), "InvalidOwnerAddress")` statement
- In the `CapxAgentFactory.sol` file, the `require( _capxAgentImpl != address(0), "InvalidAgentImplementationAddress" )` statement
- In the `CapxAgentFactory.sol` file, the `require( _capxAgentImpl.code.length > 0, "ImplementationMustBeContract" )` statement
- In the `CapxAgentFactory.sol` file, the `require( _agentParams.agentOwner != address(0), "InvalidAgentOwnerAddress" )` statement
- In the `CapxAgentFactory.sol` file, the `require( bytes(_agentParams.name).length > 0 && bytes(_agentParams.name).length <= 50, "InvalidNameLength" )` statement
- In the `CapxAgentFactory.sol` file, the `require( bytes(_agentParams.symbol).length > 0 && bytes(_agentParams.symbol).length <= 10, "InvalidSymbolLength" )` statement

- In the `CapxAgentFactory.sol` file, the `require( _isValidASCII(_agentParams.name), "InvalidNameCharacters" )` statement
- In the `CapxAgentFactory.sol` file, the `require( _isValidASCII(_agentParams.symbol), "InvalidSymbolCharacters" )` statement
- In the `CapxAgentFactory.sol` file, the `require( _agentParams.decimals >= MIN_DECIMALS && _agentParams.decimals <= MAX_DECIMALS, "InvalidDecimals" )` statement
- In the `CapxAgentFactory.sol` file, the `require( _agentParams.ecr20TotalSupply > 0 && _agentParams.ecr20TotalSupply <= MAX_TOTAL_SUPPLY, "InvalidTotalSupply" )` statement
- In the `CapxAgentFactory.sol` file, the `require(agent != address(0), "CloneDeploymentFailed")` statement
- In the `CapxAgentFactory.sol` file, the `require(limit > 0, "LimitMustBePositive")` statement

For conciseness and gas savings, consider replacing `require` and `revert` messages with custom errors.

Update: *Resolved in pull request #26 at commit 33b5355.*

N-04 Outdated Solidity Versions

Using an outdated version of Solidity can lead to vulnerabilities and unexpected behavior in contracts. As such, it is important to keep the Solidity compiler version up to date.

Throughout the codebase, multiple instances of files using outdated Solidity versions were identified:

- The `CapxAgentClone.sol` is using an outdated Solidity version (`0.8.22`).
- The `CapxAgentFactory.sol` is using an outdated Solidity version (`0.8.22`).
- The `CapxOwnableClone.sol` is using an outdated Solidity version (`0.8.22`).

Consider taking advantage of the latest Solidity version to improve the overall readability and security of the codebase. Regardless of which version of Solidity is used, consider pinning the version consistently throughout the codebase to prevent bugs due to incompatible future releases.

Update: *Resolved in pull request #27 at commit d74be45.*

N-05 Misleading Naming Between Agent Creator and Owner Fields

In the `CapxAgentFactory` contact, the `createAgent()` function, restricts execution to `onlyOwner`, so the factory owner is the deployer of all agents. However, the `capxAgentByCreator` mapping records entries under `_agentParams.agentOwner`, which refers to an address provided in the input rather than the transaction sender. This causes inconsistent terminology between the logical creator and the recorded "creator".

Consider renaming the `_agentParams.agentOwner` parameter to `_agentParams.agentCreator`.

Update: *Resolved in pull request #28 at commit 844de33.*

N-06 `initializeOwner` Lacks One-Time Initialization Guard

The `initializeOwner` function in `CapxOwnableClone.sol` can be called multiple times by the inheriting contract. This is not an issue for any contracts in scope, but could be a potential issue for any other contract that inherits the `CapxOwnableClone` contract.

To match the expected behavior, consider adding a check to prevent the `initializeOwner` function from being executed if an owner is already assigned.

Update: *Resolved in pull request #29 at commit aa244df.*

Conclusion

The Capx smart contracts are designed to enable the creation of tokens for AI applications, supporting innovative monetization models, community governance, and access control for AI services on a custom Ethereum Layer 2 network.

No critical- or high-severity vulnerabilities were identified during the review. Only minor recommendations were made, primarily related to naming clarity, code readability, and consistency. These adjustments are expected to improve developer understanding and reduce the likelihood of future integration errors.