Making Web3 Space Safer for Everyone

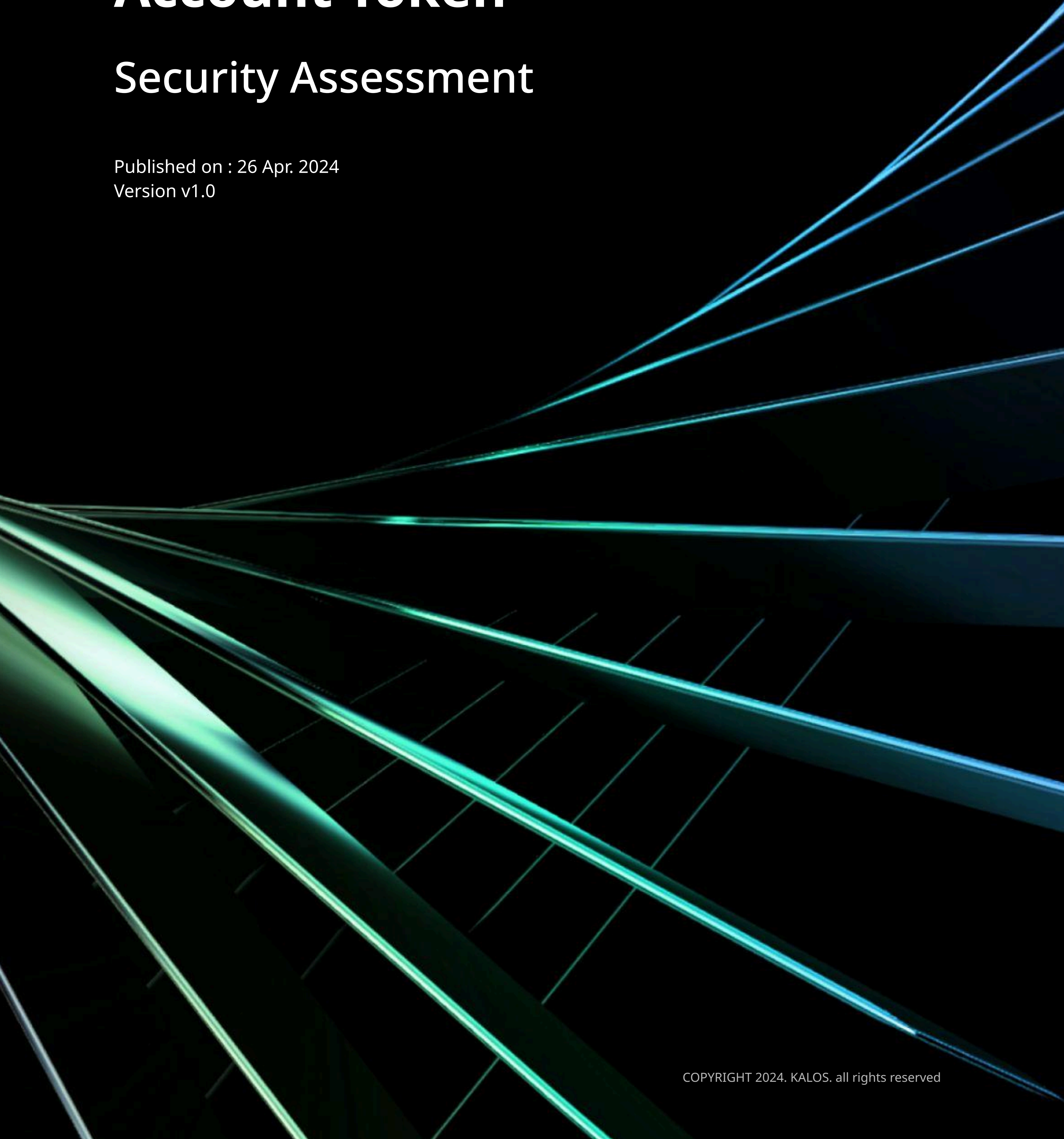
KALOS

Agent Exchange

Account Token

Security Assessment

Published on : 26 Apr. 2024
Version v1.0



Security Report Published by KALOS

v1.0 26 Apr. 2024

Auditor: Hun

Found issues

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

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 Account Token contract. 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 Account Token.
In detail, we have focused on the following

- Known Smart Contract vulnerabilities
- Unintended behavior on the minting process.
- Risk in Upgradable contract
- Asset theft from the contract
- Signature Verification Bypass

Codebase Submitted for the Audit

The codes used in this Audit can be found on GitHub (https://github.com/AgentExchange/agent-contracts/tree/094be878ad0d838bb3dc4c87e0cb67cee3237792).

Audit Timeline

Date	Event
2024/4/22	Audit Initiation (Account Token and ERC1271)
2024/4/26	Delivery of v1.0 report.

Findings

KALOS found 0 Critical, 0 High, 0 Medium, and 1 Low severity issues. There is 1 Tips issue explained that would improve the code's usability or efficiency upon modification.

Severity	Issue	Status
Low	Account with incorrect chain id returns zero address owner	(Acknowledged - v1.0)
Tips	Code Quality Improvement	(Acknowledged - v1.0)

Remarks

Only AgentAccountTokenV1is in scope. Other contracts are out of scope during this audit.
Nested EIP712 used in ERC1271 has recently been suggested. Agent Account may be affected by potential risk from it.
All issues found have not yet been completely resolved.

OVERVIEW

Protocol overview

- **AgentAccountTokenV1**

The AgentAccountTokenV1 is an ERC721 contract that creates an agent account based on ERC6551, manages a whitelist, and enables token transferability. By default, the token owner cannot transfer their tokens. Only the whitelisted owners can transfer them after transfer is enabled.

- **ERC6551**

This contract assigns token-bound smart contract accounts to NFTs. These accounts, as upgradeable wallets owned by an NFT owner, can own assets and interact with applications. Following the standard, it must implement ERC1271 signature validation. The current Solady's ERC6551 uses Solady's ERC1271 using nested EIP712.

Scope

```
├── contracts
│   ├── AgentAccountTokenV1.sol
```

Access Controls

Agent Account Token contract have the following access control mechanisms.

- ERC6551.onlyValidSigner()
- AgentAccountTokenV1.onlyOwner()

ERC6551.onlyValidSigner(): This modifier ensures that the caller of a function is a valid signer, typically an NFT owner. It utilizes the _isValidSigner function, which validates whether the msg.sender matches the current owner of NFT linked to the ERC6551 contract.

AgentAccountTokenV1.onlyOwner(): The contract owner can call *setWhitelist()* and *enableTransfer()*. *setWhitelist()* adds or removes addresses from the whitelist and *enableTransfer()* enables the token's whitelisted owners to transfer their tokens.

FINDINGS

1. Account with incorrect chain id returns zero address owner

ID: AGENT-01

Type: N/A

Severity: Low

Difficulty: Low

Impact

An account created with wrong chain id doesn't properly work since the owner() function returns a zero address.

Description

```
function safeMint(address _to, address _implementation, bytes32 _salt, uint256 _chainId)
```

[https://github.com/AgentExchange/agent-contracts/blob/main/contracts/AgentAccountTokenV1.sol#L71]

The chain id parameter of *safeMint()* is arbitrary, and a user may create an account with a different chain id.

```
uint256 private immutable _cachedChainId = block.chainid;
function owner() public view virtual returns (address result) {
    uint256 cachedChainId = _cachedChainId;
    /// @solidity memory-safe-assembly
    assembly {
        let m := mload(0x40) // Cache the free memory pointer.
        extcodecopy(address(), 0x00, 0x4d, 0x60)
        if eq(mload(0x00), cachedChainId) {
            // ...
        }
        // return zero address as owner
        mstore(0x40, m) // Restore the free memory pointer.
    }
}
```

[https://github.com/vectorized/solady/blob/b08c725360a1d109c087cc8e93f1c40e7a9f5851/src/accounts/ERC6551.sol#L107-L128

If the account's chain id different from the cached chain id (which is stored chain id at creation time), owner() of Solady's ERC6551 returns a zero address.

Therefore, a chain id should be the same as the current chain id when creating an account.

Recommendation

Delete the chain id parameter and call createAccount() with block.chainid.

2. Code Quality Improvement

ID: AGENT-02	Severity: Tips
Type: N/A	Difficulty: N/A

Description & Recommendation

These tips can improve code quality or optimize gas consumption.

1. Unnecessary storage variable

ERC6551 Registry address is used as storage variable although it won’t be changed.

Use constant or immutable registry address for gas optimization.

```
    IERC6551Registry constant public registry =
IERC6551Registry(0x000000006551c19487814612e58FE06813775758);
```

[https://github.com/AgentExchange/agent-contracts/blob/main/contracts/AgentAccountTokenV1.sol#L16]

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