SALUS SECURITY

NOV 2024

CODE SECURITY ASSESSMENT

WABA

Overview

Project Summary

- Name: WABA - ALSC
- Platform: EVM-compatible chains
- Language: Solidity
- Address:
  - Proxy contract:0x9481F6888a568C1C2C29F998d98A99F4c395848d
  - Implementation contract: 0x244E3b90AD86914a079ba2406cb09C2287f94E25
- Audit Range: See Appendix - 1

Project Dashboard

Application Summary

Name	WABA - ALSC
Version	v2
Type	Solidity
Dates	Nov 12 2024
Logs	Oct 09 2024; Nov 12 2024

Vulnerability Summary

Total High-Severity issues	1
Total Medium-Severity issues	1
Total Low-Severity issues	2
Total informational issues	1
Total	5

Contact

E-mail: support@salusec.io

Risk Level Description

High Risk	The issue puts a large number of users’ sensitive information at risk, or is reasonably likely to lead to catastrophic impact for clients’ reputations or serious financial implications for clients and users.
Medium Risk	The issue puts a subset of users’ sensitive information at risk, would be detrimental to the client’s reputation if exploited, or is reasonably likely to lead to a moderate financial impact.
Low Risk	The risk is relatively small and could not be exploited on a recurring basis, or is a risk that the client has indicated is low impact in view of the client’s business circumstances.
Informational	The issue does not pose an immediate risk, but is relevant to security best practices or defense in depth.

Content

Introduction

1.1 About SALUS

At Salus Security, we are in the business of trust.

We are dedicated to tackling the toughest security challenges facing the industry today. By building foundational trust in technology and infrastructure through security, we help clients to lead their respective industries and unlock their full Web3 potential.

Our team of security experts employ industry-leading proof-of-concept (PoC) methodology for demonstrating smart contract vulnerabilities, coupled with advanced red teaming capabilities and a stereoscopic vulnerability detection service, to deliver comprehensive security assessments that allow clients to stay ahead of the curve.

In addition to smart contract audits and red teaming, our Rapid Detection Service for smart contracts aims to make security accessible to all. This high calibre, yet cost-efficient, security tool has been designed to support a wide range of business needs including investment due diligence, security and code quality assessments, and code optimisation.

We are reachable on Telegram (https://t.me/salusec), Twitter (https://twitter.com/salus_sec), or Email (support@salusec.io).

1.2 Audit Breakdown

The objective was to evaluate the repository for security-related issues, code quality, and adherence to specifications and best practices. Possible issues we looked for included (but are not limited to):

- Risky external calls
- Integer overflow/underflow
- Transaction-ordering dependence
- Timestamp dependence
- Access control
- Call stack limits and mishandled exceptions
- Number rounding errors
- Centralization of power
- Logical oversights and denial of service
- Business logic specification
- Code clones, functionality duplication

1.3 Disclaimer

Note that this security audit is not designed to replace functional tests required before any software release and does not give any warranties on finding all possible security issues with the given smart contract(s) or blockchain software, i.e., the evaluation result does not guarantee the nonexistence of any further findings of security issues.

Findings

2.1 Summary of Findings

ID	Title	Severity	Category	Status
1	Signature reuse in the NFT coin process	High	Business Logic	Resolved
2	Lack of slippage protection for buyers	Medium	Business Logic	Acknowledged
3	Implementation contract could be initialized by everyone	Low	Business Logic	Acknowledged
4	Missing events for functions that change critical state	Low	Logging	Acknowledged
5	Missing __ERC721Enumerable_init() call in initialize function	Informational	Code Quality	Acknowledged

2.2 Notable Findings

Significant flaws that impact system confidentiality, integrity, or availability are listed below.

1. Signature reuse in the NFT coin process	
Severity: High	Category: Business Logic
Target: - ALSC.sol	

Description

The ALSC contract contains a critical vulnerability in its buy function that allows for the reuse of signatures. This function is responsible for minting new NFTs based on a signature provided by an authorized signer. However, the contract fails to check whether a signature has been previously used before processing the minting request.

ALSC.sol:L26-L37

```
function buy(bytes memory signature) public payable {
    require(state, "not open");
    require(currentIndex < maxSupply, "exceed max supply");
    address signer = ECDSAUpgradeable.recover(
        getMessageHash(msg.sender),
        signature
    );
    require(signers[signer], "Signer is not valid");
    sigUsed[signature] = true;
    refundIfOver(price);
    _mint(msg.sender, currentIndex);
}
```

While the contract does mark signatures as used after successful minting (sigUsed[signature] = true), it does not verify if the signature has been used before processing the request. This oversight allows an attacker to reuse a valid signature multiple times, potentially minting more NFTs than intended or authorized.

Recommendation

It is recommended to implement a check to ensure that each signature is only used once.

Status

The team has resolved this issue in commit f21b7335.

The team has resolved this issue in the Implementation contract 0x7771b1FBE4F1211c36069D02a572CB07a0d3908C.

2. Lack of slippage protection for buyers	
Severity: Medium	Category: Business Logic
Target: - ALSC.sol	

Description

The ALSC contract allows the contract owner to front-run buyers and unfairly increase the price of NFTs just before a purchase is completed. This is possible due to two key factors:

1. ALSC.sol:L45-L47

```
function setPrice(uint256 price_) public onlyOwner {
    price = price_;
}
```

The owner has the ability to change the price at any time.

2. ALSC.sol:L26-L37

```
function buy(bytes memory signature) public payable {
    // ... (other checks)
    refundIfOver(price);
    _mint(msg.sender, currentIndex);
}
```

The buy function uses the current price state variable without any protection against last-minute changes.

Attach Scenario

This setup allows the following attack scenario:

1. A buyer submits a transaction to purchase an NFT at the current price.
2. The owner sees this transaction in the mempool.
3. The owner quickly submits a transaction to increase the price, with a higher gas price to ensure it's processed first.
4. If the buyer's transaction included more ETH than the original price (common to account for potential slight price increases), the purchase would still go through at the new, higher price.

As a result, buyers may end up paying significantly more than they intended.

Recommendation

It is recommended to include the expected price in the signed message for the buy function.

Status

This issue has been acknowledged by the team.

3. Implementation contract could be initialized by everyone	
Severity: Low	Category: Business Logic
Target: - ALSC.sol	

Description

According to OpenZeppelin, the implementation contract should not be left uninitialized.

An uninitialized implementation contract can be taken over by an attacker, which may impact the proxy. There is nothing preventing the attacker from calling the `` `initialize()` `` function in ALSC's implementation contract.

Recommendation

To prevent the implementation contract from being used, consider invoking the _disableInitializers function in the constructor of the `` `ALSC` `` contract to automatically lock it when it is deployed.

4. Missing events for functions that change critical state	
Severity: Low	Category: Logging
Target: - ALSC.sol	

Description

Events allow capturing the changed parameters so that off-chain tools/interfaces can register such changes that allow users to evaluate them. Missing events do not promote transparency and if such changes immediately affect users' perception of fairness or trustworthiness, they could exit the protocol causing a reduction in protocol users.

In the `ALSC.sol`, events are missing in the following functions:

- setBaseURI()
- setPrice()
- setMaxSupply()
- open()
- close()
- setSigner()

Recommendation

It is recommended to emit events for critical state changes.

Status

This issue has been acknowledged by the team.

2.3 Informational Findings

5. Missing __ERC721Enumerable_init() call in initialize function	
Severity: Informational	Category: Code Quality
Target: - ALSC.sol	

Description

The ALSC contract inherits from ERC721EnumerableUpgradeable but fails to properly initialize it. In the initialize() function, only __ERC721_init() is called, omitting the __ERC721Enumerable_init() call.

ALSC.sol:L19-L24

```
function initialize() public initializer {
    __ERC721_init("AI Link Smart Charger", "WABA ALSC");
    __Ownable_init();
    maxSupply = 500;
    price = 0.05 ether;
}
```

Recommendation

Modify the initialize() function to include the __ERC721Enumerable_init() call.

Status

This issue has been acknowledged by the team.

Appendix

Appendix 1 - Files in Scope

This audit covered files at the following address:

- Proxy contract:0x9481F6888a568C1C2C29F998d98A99F4c395848d
- Implementation contract:
  0x244E3b90AD86914a079ba2406cb09C2287f94E25