SALUS SECURITY

JUL 2024

CODE SECURITY ASSESSMENT

MEME

Overview

Project Summary

- Name: Meme
- Platform: EVM-compatible chains
- Language: Solidity
- Audit Range: See Appendix - 1

Project Dashboard

Application Summary

Name	Meme
Version	v2
Type	Solidity
Dates	Jul 02 2024
Logs	Jul 01 2024; Jul 02 2024

Vulnerability Summary

Total High-Severity issues	0
Total Medium-Severity issues	2
Total Low-Severity issues	5
Total informational issues	1
Total	8

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 **4**
1.1 About SALUS 4
1.2 Audit Breakdown 4
1.3 Disclaimer 4
Findings **5**
2.1 Summary of Findings 5
2.2 Notable Findings 6
1. Possible replay attacks during createToken() 6
2. Centralization risk 7
3. Redundant and dangerous permission functions 8
4. Implementation contract could be initialized by everyone 9
5. Lack of upper limit for fees in _launchFee() 10
6. Different tokens may have the same symbol 11
7. Missing events for functions that change critical state 12
2.3 Informational Findings 13
8. The createToken() will reverts in some cases 13
Appendix **14**
Appendix 1 - Files in Scope 14

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	Possible replay attacks during createToken()	Medium	Cryptography	Acknowledged
2	Centralization risk	Medium	Centralization	Acknowledged
3	Redundant and dangerous permission functions	Low	Business Logic	Acknowledged
4	Implementation contract could be initialized by everyone	Low	Business Logic	Acknowledged
5	Lack of upper limit for fees in _launchFee()	Low	Centralization	Acknowledged
6	Different tokens may have the same symbol	Low	Business logic	Acknowledged
7	Missing events for functions that change critical state	Low	Logging	Acknowledged
8	The createToken() will reverts in some cases	Informational	Numerics	Acknowledged

2.2 Notable Findings

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

1. Possible replay attacks during createToken()	
Severity: Medium	Category: Cryptography
Target: - TokenManager.sol	

Description

The `createToken()` function allows the creation of a new token by a third-party who has a signature from `signer()`.

However, the `chainId` field is missing in the signature. This means that a user with the signature can use it for verification on multiple chains.

Even worse, once `TokenManager.sol` is deployed on multiple chains, Alice, who holds the signature for requestId 1 on chain A, can front-run Bob's transaction with the same requestId on chain B.

Recommendation

Consider adding the `chainId` to the signature field.

Status

This issue has been acknowledged by the team. The team has claimed that the `chainId` field will be added when deployed on other chains.

2. Centralization risk	
Severity: Medium	Category: Centralization
Target: - TokenManager.sol	

Description

In the `` `TokenManager.sol` ``, there is a privileged owner role. This role has the ability to withdraw all ETH.

TokenManager.sol:L437-L439

```
function withdrawEth(address to, uint256 amount) public onlyOwner {
    Address.sendValue(payable(to), amount);
}
```

If an attacker were to gain access to the private keys associated with this role, they could cause significant damage to the project.

Recommendation

We recommend transferring privileged accounts to multi-sig accounts with timelock governors for enhanced security. This ensures that no single person has full control over the accounts and that any changes must be authorized by multiple parties.

Status

This issue has been acknowledged by the team. The team has stated that they will move owner permissions to a multi-signature address after deployment.

3. Redundant and dangerous permission functions	
Severity: Low	Category: Business Logic
Target: - BasicAccessControl.sol	

Description

BasicAccessControl.sol:L53-L67

```
function grantDeployer(address account) public {
    grantRole(ROLE_DEPLOYER, account);
}

function revokeDeployer(address account) public {
    revokeRole(ROLE_DEPLOYER, account);
}

function grantOperator(address account) public {
    grantRole(ROLE_OPERATOR, account);
}

function revokeOperator(address account) public {
    revokeRole(ROLE_OPERATOR, account);
}
```

In the `` `BasicAccessControl.sol` ``, there are four public functions provided regarding permission operations.

@openzeppelin/contracts@4.9.6/access/AccessControl.sol

```
function grantRole(bytes32 role, address account) public virtual override
onlyRole(getRoleAdmin(role)) {
    _grantRole(role, account);
}
```

However, the functions called by `` `BasicAccessControl.sol` `` have strict access controls requiring the caller to be a role administrator.

This means that, under proper rights management, those four functions in `` `BasicAccessControl.sol` `` will never be successfully called.

Recommendation

Consider implementing access control in `` `BasicAccessControl.sol` `` and directly calling the underlying function `` `_grantRole()` ``.

Status

This issue has been acknowledged by the team.

4. Implementation contract could be initialized by everyone	
Severity: Low	Category: Business Logic
Target: - TokenManager.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()` `` in `` `TokenManager.sol` ``'s implementation contract.

Recommendation

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

Status

This issue has been acknowledged by the team.

5. Lack of upper limit for fees in _launchFee()	
Severity: Low	Category: Centralization
Target: - TokenManager.sol	

Description

The `_launchFee()` function is used to modify the value of the launch fee.
TokenManager.sol:L131-L133

```
function setLaunchFee(uint256 v) public onlyOperator {
    _launchFee = v;
}
```

However, the function has no upper limit on fees. If the fee setter's private key is compromised, the attacker can set launch fee to type(uint256).max, preventing users from performing createToken operations.

Recommendation

It is recommended to include a reasonable upper limit for launch fee in _launchFee().

Status

This issue has been acknowledged by the team.

6. Different tokens may have the same symbol	
Severity: Low	Category: Business Logic
Target: - TokenManager.sol	

Description

TokenManager.sol:L401-L402

```
function _createToken(
   ...
) internal returns (address) {
    ...
    Token token = new Token(name, symbol, totalSupply);
    ...
}
```

When creating tokens, the protocol allows tokens to be created with the same name and symbol.

If there are two tokens with identical symbols, users will find it difficult to distinguish between them. This could significantly degrade user experience and even lead to user attrition.

Recommendation

Consider enforcing unique names and symbols for tokens created through TokenManager.

Status

This issue has been acknowledged by the team.

7. Missing events for functions that change critical state	
Severity: Low	Category: Logging
Target: - TokenManager.sol - UseSigner.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 `` `TokenManager.sol` ``, the following functions are lacking events:

- setLaunchFee()
- setTradeFeeRate()
- setMinTradeFee()
- setFeeRecipient()
- suspendTrading()
- stopTrade()
- withdrawEth()

In `` `UseSigner.sol` `` the function `` `setSigner()` `` is lacking an event.

Recommendation

It is recommended to emit events for critical state changes.

Status

This issue has been acknowledged by the team.

2.3 Informational Findings

8. The createToken() will reverts in some cases	
Severity: Informational	Category: Numerics
Target: - TokenManager.sol	

Description

The function `` `_createToken()` `` does a lot of checking to avoid unexpected revert.
But there are still some cases where calling of `` `_createToken()` `` will revert.

For example:
- total supply: 100e18;
- max offer: 100e18/2 + 1.

Recommendation

Despite the fact that only the owner can access the createToken() function, it is better to implement the minimum ratio `maxOffer/totalSupply to avoid reverts.

Status

This issue has been acknowledged by the team.

Appendix

Appendix 1 - Files in Scope

The audit covered the following files:

File	SHA-1 hash
TokenManager.sol	c32143078089efe542fdc530e28247e0b4c201e3
OwnableUpgradeable.sol	4f23c627bb7f32e70be5e14f9c9808603d69b7d3
BasicAccessControl.sol	9c1463b2c18bc95400a40fca08f674b40771e487
Deployer.sol	d4e9cada1008d8ca5a04429041772ca17898d4fc
UseSigner.sol	4948ca6bafafc0975059ac1e6214770a18b7eac8
SignatureVerifier.sol	eaefaa876fc5ce058b2bcb42883bcb74b289b15d
ContextUpgradeable.sol	fe1ee89a033e78db542431803036beeac3f810b7
RequestTracing.sol	358f2735864a669feccdf1923970bf8a0cde942b