SALUS SECURITY

OCT 2023

CODE SECURITY ASSESSMENT

UNYX TECH

Overview

Project Summary

- Name: Unyx Tech - Tomo
- Version: commit f75e193
- Platform: EVM-compatible chains
- Language: Solidity
- Repository:
    - https://github.com/UnyxTech/tomo-contracts
- Audit Range: See Appendix - 1

Project Dashboard

Application Summary

Name	Unyx Tech - Tomo
Version	v3
Type	Solidity
Dates	Oct 12 2023
Logs	Oct 11 2023; Oct 12 2023; Oct 12 2023

Vulnerability Summary

Total High-Severity issues	0
Total Medium-Severity issues	1
Total Low-Severity issues	4
Total informational issues	3
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

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	Centralization risk	Medium	Centralization	Acknowledged
2	Surplus of msg.value will be locked in the contract	Low	Business Logic	Acknowledged
3	Impairment loss of votes/passes if KOLs rename their accounts	Low	Business Logic	Acknowledged
4	Signature replay in the buyVotePass() function	Low	Business Logic	Acknowledged
5	Missing events for functions that change critical state	Low	Logging	Acknowledged
6	Lack of indexed parameters in events	Informational	Logging	Acknowledged
7	Missing zero address checks	Informational	Data Validation	Acknowledged
8	Gas optimization suggestions	Informational	Gas Optimization	Acknowledged

2.2 Notable Findings

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

1. Centralization risk	
Severity: Medium	Category: Centralization
Target: - src/Tomo.sol	

Description

There is a privileged owner role in the Tomo contract. The owner of the Tomo contract can manage the signers, set the recipient of the protocol fee, and modify critical settings, such as feePercent, merkleRoot.

Should the owner's private key be compromised, an attacker could set the protocolFeeTo to an address he controls and set feePercent configurations to the maximum.

If the privileged accounts are plain EOA accounts, this can be worrisome and pose a risk to the other users.

Recommendation

It is recommended to transfer 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.

2. Surplus of msg.value will be locked in the contract	
Severity: Low	Category: Business Logic
Target: - src/Tomo.sol	

Description

In the buyVotePass() function, there is only a check that msg.value should be greater than or equal to the params.value. If a user mistakenly sends more msg.value than params.value, the surplus will not be returned to the user and will be locked in the contract forever.

src/Tomo.sol:L318-L324

```
VotePass storage vp = votePasses[subject];
params.price = getPrice(vp.totalSupply, amount);
params.protocolFee = params.price * protocolFeePercent / 1 ether;
params.subjectFee = params.price * subjectFeePercent / 1 ether;
params.rewardFee = params.price * rewardFeePercent / 1 ether;
params.value = params.price + params.protocolFee + params.subjectFee + params.rewardFee;
require(msg.value >= params.value, "Insufficient payment");
```

Recommendation

Consider returning the surplus ether to the user or changing `msg.value >= params.value` to `msg.value == params.value`.

Status

This issue has been acknowledged by the team.

3. Impairment loss of votes/passes if KOLs rename their accounts	
Severity: Low	Category: Business Logic
Target: - src/Tomo.sol	

Description

The trading of votes/passes and distribution of rewards are based on the subject, which is derived from the platform and the name of the KOL. If a KOL renames his account on social networks, all previous votes/passes can not be transferred to the new subject.

Recommendation

Consider adding a privileged function to transfer funds under the subject or handling this situation off-chain.

Status

This issue has been acknowledged by the team. A user's handle in Tomo can not be changed after registration, and the user is associated with accounts on other social networks through OpenID.

4. Signature replay in the buyVotePass() function	
Severity: Low	Category: Business Logic
Target: - src/Tomo.sol	

Description

Users can buy a certain number of votes/passes based on signatures signed by all authorized signers. However, due to lack of nonce control, users can buy more votes/passes by replaying valid signatures.

src/Tomo.sol:L309-L350

```
function buyVotePass(
    ...
) external payable nonReentrant {
    recover(buildBuySeparator(subject, msg.sender, amount), v, r, s);
    ...
}
```

src/Tomo.sol:L549-L567

```
function recover(
       bytes32 hash,
       uint8[] calldata v,
       bytes32[] calldata r,
       bytes32[] calldata s
) public view returns (bool) {
       uint256 length = signers.length;
       require(length > 0 && length == v.length && length == r.length && length ==
s.length, "Invalid signature length");
       address[] memory signatures = new address[](length);
       for (uint256 i = 0; i < length; i++) {
       address signer = ecrecover(hash, v[i], r[i], s[i]);
       require(authorized[signer], "Invalid signer");
       for (uint256 j = 0; j < i; j++) {
              require(signatures[j] != signer, "Duplicated");
       }
       signatures[i] = signer;
       }
       return true;
}
```

Recommendation

If repeat purchases via signature replay are not as expected, consider including a nonce for replay protection.

Status

This issue has been acknowledged by the team.

5. Missing events for functions that change critical state	
Severity: Low	Category: Logging
Target: - src/Tomo.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 Tomo contract, events are lacking in the privileged setter functions (e.g. setProtocolFeeTo(), setProtocolFeePercent()).

Recommendation

It is recommended to emit events for critical state changes.

Status

This issue has been acknowledged by the team.

2.3 Informational Findings

6. Lack of indexed parameters in events	
Severity: Informational	Category: Logging
Target: - src/Tomo.sol	

Description

In the Tomo contract, there are no indexed event parameters.

src/Tomo.sol:L269-L271

```
event SignerAdded(address sender, address account);
event SignerRemoved(address sender, address account);
event BindSubject(uint256 eventIndex, uint256 ts, bytes32 subject, address owner);
```

Recommendation

Consider indexing important event parameters to improve off-chain services' ability to search and filter for specific events.

Status

This issue has been acknowledged by the team.

7. Missing zero address checks	
Severity: Informational	Category: Data Validation
Target: - src/Tomo.sol	

Description

It is considered a security best practice to verify addresses against the zero address during initialization or setting. However, this precautionary step is absent for the address variable account.

src/Tomo.sol:L491-L497

```
function addSigner(address account) external onlyOwner {
       require(!authorized[account], "Not reentrant");
       indexes[account] = signers.length;
       authorized[account] = true;
       signers.push(account);
       emit SignerAdded(msg.sender, account);
}
```

Meanwhile, in the recover() function, the address returned by ecrecover() will be verified if it is authorized. When the signature is invalid, ecrecover() will return address(0). Thus, if there is only one zero-address signer, the signature verification can be bypassed.

src/Tomo.sol:L559-L560

```
address signer = ecrecover(hash, v[i], r[i], s[i]);
require(authorized[signer], "Invalid signer");
```

Recommendation

Consider adding zero address checks for the address variable account.

Status

This issue has been acknowledged by the team.

8. Gas optimization suggestions	
Severity: Informational	Category: Gas Optimization
Target: - src/Tomo.sol	

Description

src/Tomo.sol:L208

```
bytes32 public DOMAIN_SEPARATOR;
```

DOMAIN_SEPARATOR is only set in the constructor and can't be updated after deployment of the contract. So, it can be declared immutable to save gas.

Recommendation

Consider using the immutable modifier for the state variable DOMAIN_SEPARATOR.

Status

This issue has been acknowledged by the team.

Appendix

Appendix 1 - Files in Scope

This audit covered the following files in commit f75e193:

File	SHA-1 hash
src/Tomo.sol	f69e435b34f81cea87b638b5970f2ff173a1d1a0