Smart Contract Security Audit Report

A7A5 Token Audit

1. Contents

2. General Information

This report contains information about the results of the security audit of the A7A5 (hereafter referred to as "Customer") smart contracts, conducted by Decurity in the period from 21/01/2025 to 23/01/2025.

2.1. Introduction

Tasks solved during the work are:

- Review the protocol design and the usage of 3rd party dependencies,
- Audit the contracts implementation,
- Develop the recommendations and suggestions to improve the security of the contracts.

2.2. Scope of Work

The audit scope included the contracts in the following repository: https://github.com/a7a5-defi/a7a5. Initial review was done for the commit 3bfbe2.

The following contracts have been tested:

- contracts/A7A5.sol

2.3. Threat Model

The assessment presumes actions of an intruder who might have capabilities of any role (an external user, token owner, token service owner, a contract). The centralization risks have not been considered upon the request of the Customer.

The main possible threat actors are:

- User,
- Protocol owner,
- Liquidity Token owner/contract.

The table below contains sample attacks that malicious attackers might carry out.

Table. Theoretically possible attacks

Attack	Actor
Contract code or data hijacking *Deploying a malicious contract or submitting malicious data*	Contract owner Token owner
Financial fraud *A malicious manipulation of the business logic and balances, such as a re-entrancy attack or a flash loan attack*	Anyone
Attacks on implementation *Exploiting the weaknesses in the compiler or the runtime of the smart contracts*	Anyone

2.4. Weakness Scoring

An expert evaluation scores the findings in this report, an impact of each vulnerability is calculated based on its ease of exploitation (based on the industry practice and our experience) and severity (for the considered threats).

2.5. Disclaimer

Due to the intrinsic nature of the software and vulnerabilities and the changing threat landscape, it cannot be generally guaranteed that a certain security property of a program holds.

Therefore, this report is provided “as is” and is not a guarantee that the analyzed system does not contain any other security weaknesses or vulnerabilities. Furthermore, this report is not an endorsement of the Customer’s project, nor is it an investment advice.

That being said, Decurity exercises best effort to perform their contractual obligations and follow the industry methodologies to discover as many weaknesses as possible and maximize the audit coverage using the limited resources.

3. Summary

As a result of this work, we have discovered a single medium security issue. The other suggestions included fixing the low-risk issues and some best practices (see Security Process Improvement). These vulnerabilities have been addressed and thoroughly re-tested as part of our process.

3.1. Suggestions

The table below contains the discovered issues, their risk level, and their status as of February 3, 2025.

Table. Discovered weaknesses

Issue	Contract	Risk Level	Status
Inconsistent approves	contracts/A7A5.sol	Medium	Fixed
Unintended token burning	contracts/A7A5.sol	Low	Fixed
Incorrect event value during transfer	contracts/A7A5.sol	Low	Fixed
Lack of zero checks	contracts/A7A5.sol	Low	Fixed
Lack of event emitting	contracts/A7A5.sol	Low	Fixed
Rounding issues in token transfer	contracts/A7A5.sol	Info	Fixed
Fees can be bypassed	contracts/A7A5.sol	Info	Acknowledged
Redundant restriction in approve	contracts/A7A5.sol	Info	Fixed
Typo in the revert message	contracts/A7A5.sol	Info	Fixed

4. General Recommendations

This section contains general recommendations on how to improve overall security level.

The Findings section contains technical recommendations for each discovered issue.

4.1. Security Process Improvement

The following is a brief long-term action plan to mitigate further weaknesses and bring the product security to a higher level:

- Keep the whitepaper and documentation updated to make it consistent with the implementation and the intended use cases of the system,
- Perform regular audits for all the new contracts and updates,
- Ensure the secure off-chain storage and processing of the credentials (e.g. the privileged private keys),
- Launch a public bug bounty campaign for the contracts.

5. Findings

5.1. Inconsistent approves

Risk Level: Medium

Status: Fixed in the commit f734b1.

Contracts:

- contracts/A7A5.sol

Description:

The `transferFrom` and `transferScaledFrom` functions contain inconsistencies in how allowances are deducted and compared, which could lead to transaction reverts.

The `transferFrom` Function:

- The function compares the allowance against `_value` to ensure the spender has sufficient allowance (`require(allowance_ >= _value, "allowance exceeded");`).
- However, it deducts `scaledAmount` (a scaled version of `_value`) from the allowance (`_allowances[_from][msg.sender] -= scaledAmount;`).
- If `scaledAmount` is greater than `_value`, the transaction will revert because the allowance check (`require(allowance_ >= _value)`) will pass, but the deduction of `scaledAmount` will attempt to subtract a larger value than the allowance.

```
function transferFrom(
      address _from,
      address _to,
      uint256 _value
  ) public whenNotPaused notBlacklisted(_from) returns (bool) {
      uint256 allowance_ = _allowances[_from][msg.sender];
      require(allowance_ >= _value, "allowance exceeded");

      uint256 scaledAmount = getScaledAmount(_value);
      uint256 fee = (scaledAmount * basisPointsRate) / FEE_PRECISION;
      if (allowance_ < type(uint256).max) {
          _allowances[_from][msg.sender] -= scaledAmount; //  @audit should
deduct "value"
      }
      _transferShares(_from, _to, scaledAmount - fee);
```

```
        if (fee > 0) {
            _transferShares(_from, owner, fee);
            emit Transfer(_from, owner, _value);
        }
        emit Transfer(_from, _to, _value);
        return true;
    }
```

The `transferScaledFrom` Function:

- The function compares the allowance against `liquidityAmount` (`require(allowance_ >= liquidityAmount, "allowance exceeded");`).
- However, it deducts `_value` from the allowance (`_allowances[_from][msg.sender] -= _value;`).
- If `_value` is greater than `liquidityAmount`, the transaction will revert because the allowance check (`require(allowance_ >= liquidityAmount)`) will pass, but the deduction of `_value` will attempt to subtract a larger value than the allowance.

```
function transferScaledFrom(
        address _from,
        address _to,
        uint256 _value
    ) public whenNotPaused notBlacklisted(_from) returns (bool) {
        uint256 fee = (_value * basisPointsRate) / FEE_PRECISION;
        uint256 allowance_ = _allowances[_from][msg.sender];
        uint256 liquidityAmount = getLiquidityAmount(_value);
        require(allowance_ >= liquidityAmount, "allowance exceeded");
        if (allowance_ < type(uint256).max) {
            _allowances[_from][msg.sender] -= _value; // @audit should deduct
"liquidityAmount"
        }
        _transferShares(_from, _to, _value - fee);
        if (fee > 0) {
            _transferShares(_from, owner, fee);
        }
        return true;
    }
```

Remediation:

Consider deducting correct value.

5.2. Unintended token burning

Risk Level: Low

Status: Fixed in the commit f734b1.

Contracts:

- contracts/A7A5.sol

Description:

The contract logic specifies that only the owner can burn tokens using the `burn` function. However, there is a way that allows unauthorized token burning through a combination of front-running and exploiting the `destroyBlackFunds` and `transfer` functions.

1. A user front-runs a `destroyBlackFunds` transaction to transfer tokens to a blacklisted user.
2. The compliance role calls `destroyBlackFunds` to burn the tokens held by the blacklisted user.
3. As a result, user's tokens are burned without the owner's explicit authorization, bypassing the restriction in the `burn` function.

```
function transfer(
      address _to,
      uint256 _value
  ) public whenNotPaused notBlacklisted(msg.sender) returns (bool) {
      // @audit should check notBlacklisted(to) (burn)
      uint256 scaledAmount = getScaledAmount(_value);
      uint256 fee = (scaledAmount * basisPointsRate) / FEE_PRECISION;
      _transferShares(msg.sender, _to, scaledAmount - fee);
      if (fee > 0) {
          _transferShares(msg.sender, owner, fee);
          emit Transfer(msg.sender, owner, _value);
      }
      emit Transfer(msg.sender, _to, _value);
      return true;
  }
```

Remediation:

Consider adding modifier`notBlacklisted(_to)` for transfer functions.

5.3. Incorrect event value during transfer

Risk Level: Low

Status: Fixed in the commit f734b1.

Contracts:

- contracts/A7A5.sol

Description:

The `Transfer` events emitted in the `transfer` and `transferFrom` functions are incorrectly using `_value` instead of the actual transferred fee amount in the case of fee deduction. This leads to inaccurate event data, which can cause confusion for off-chain services or tools relying on event logs.

Remediation:

Corrected code for `transfer` function:

```
if (fee > 0) {
    _transferShares(msg.sender, owner, fee);
    emit Transfer(msg.sender, owner, fee); // Emit the correct fee amount
}
```

Corrected code for `transferFrom` function:

```
if (fee > 0) {
    _transferShares(_from, owner, fee);
    emit Transfer(_from, owner, fee); // Emit the correct fee amount
}
```

5.4. Lack of zero checks

Risk Level: Low

Status: Fixed in the commit f734b1.

Contracts:

- contracts/A7A5.sol

Description:

The code contains multiple instances where zero address checks and validations are missing.

Loss of Control

If the zero address is assigned to critical roles (e.g., `owner`, `compliance`, `accountant`), the contract may become unmanageable, as these roles have privileged access to key functions.

```
constructor(
        string memory name_,
        string memory symbol_,
        uint8 decimals_,
        address owner_,
        address compliance_,
        address accountant_
    ) {
        // @audit missing zero checks
        _name = name_;
        _symbol = symbol_;
        _decimals = decimals_;
        owner = owner_;
        compliance = compliance_;
        accountant = accountant_;
    }
```

Token Blocking

Transferring shares to the zero address could result in tokens being "burned" without `totalSupply` changing, which may not be the intended behavior. This could lead to a loss of tokens and disrupt the token economy.

```
function _transferShares(
        address _from,
        address _to,
        uint256 _sharesAmount
    ) internal returns (bool) {
        // @audit no "to" zero check
        require(
            _shares[_from] >= _sharesAmount,
            "not enough shares for transfer"
        );
        _shares[_from] -= _sharesAmount;
        _shares[_to] += _sharesAmount;

        return true;
    }
```

Remediation:

Consider validating that addresses are not equals to zero.

5.5. Lack of event emitting

Risk Level: Low

Status: Fixed in the commit f734b1.

Contracts:

- contracts/A7A5.sol

Description:

The `transferScaled()` and `transferScaledFrom()` functions do not emit an event after performing a token transfer. In Solidity, it is considered best practice to emit an event whenever a state-changing action occurs, particularly for token transfers.

Remediation:

To ensure transparency and improve traceability, consider emitting a `Transfer` event after the transfer operation within both functions.

5.6. Rounding issues in token transfer

Risk Level: Info

Status: Fixed in the commit f734b1.

Contracts:

- contracts/A7A5.sol

Description:

The transferred shares may be several wei less than intended amount due to rounding down in integer division. As a result, the recipient may receive a few wei less than expected during token transfers.

```
    function balanceOf(address account) public view override returns (uint256)
{
        return getLiquidityAmount(_shares[account]);
    }

    function getLiquidityAmount(uint256 shares) public view returns (uint256)
{
        if (_totalSupply == 0) {
            return 0;
        }
        return (shares * _totalLiquidity) / _totalSupply;
    }
```

The amount of possible loss will be approximately equals to `(x - rate - 1)`. The detailed explanation may be found in this issue.

Also after fixing the allowance issue in the `transferScaledFrom` function, a new vulnerability arises due to precision loss in the `getLiquidityAmount` function. This vulnerability could allow an attacker to steal `1 wei` of shares from another user's balance. Here's how the issue occurs:

```
function transferScaledFrom(
        address _from,
        address _to,
        uint256 _value
    ) public whenNotPaused notBlacklisted(_from) returns (bool) {
        uint256 fee = (_value * basisPointsRate) / FEE_PRECISION;
        uint256 allowance_ = _allowances[_from][msg.sender];
        uint256 liquidityAmount = getLiquidityAmount(_value);
        require(allowance_ >= liquidityAmount, "allowance exceeded");
        if (allowance_ < type(uint256).max) {
-           _allowances[_from][msg.sender] -= _value;
+           _allowances[_from][msg.sender] -= liquidityAmount;
        }
        _transferShares(_from, _to, _value - fee); // @audit-issue if
totalLiquidity < totalSupply it is possible to steal 1 wei shares from someone
balance
        if (fee > 0) {
            _transferShares(_from, owner, fee);
        }

        return true;
    }
```

4. `getLiquidityAmount` **Precision Loss:**
   - The `getLiquidityAmount` function calculates the liquidity amount based on the ratio of `_totalLiquidity` to `_totalSupply`.
   - If `_totalLiquidity < _totalSupply`, and the input `shares` is very small (e.g., `1 wei`), the result of `(shares * _totalLiquidity) / _totalSupply` may round down to `0` due to integer division.
   - This means that for small values of `shares`, `getLiquidityAmount` may return `0`, even though `shares` is non-zero.
5. **Exploiting Precision Loss in** `transferScaledFrom`**:**

- The `transferScaledFrom` function uses `getLiquidityAmount` to calculate the `liquidityAmount` and checks if the allowance is sufficient.
- If `getLiquidityAmount` returns `0` for a small `_value` (e.g., `1 wei`), the allowance check (`require(allowance_ >= liquidityAmount, "allowance exceeded");`) will pass, even if the actual `_value` is non-zero.
- The function then deducts `liquidityAmount` (which is `0`) from the allowance, but transfers `_value - fee` shares, which could be `1 wei`.

Remediation:

Take into account that this issue is present in your code and make a recommendation of querying token balances before and after every transfer and transferring the difference between them instead.

To fix the second issue consider checking that a `_value` passed to the `transferScaledFrom()` is not equals to zero after converting it to the liquidity amount via `getLiquidityAmount()`.

References:

- https://docs.lido.fi/guides/lido-tokens-integration-guide#1-2-wei-corner-case

5.7. Fees can be bypassed

Risk Level: Info

Status: Acknowledged

Contracts:

- contracts/A7A5.sol

Description:

Fees can be bypassed by transferring in small fragments.

The contract implements a fee mechanism where each transfer is charged a percentage fee based on the basis points rate. However, due to integer division in the fee calculation, transfers of small amounts may result in zero fees, allowing users to bypass the fee mechanism entirely.

Consider a transfer of 1000 tokens with a 0.1% fee (10 basis points):

- Single transfer of 1000 tokens:

Fee = (1000 * 10) / 10000 = 1 token

- Split into 10 transfers of 100 tokens each:

Fee per transfer = (100 * 10) / 10000 = 0 tokens (rounds down)

This attack may be useful on Tron blockchain.

Remediation:

Consider adding a minimal fee value.

5.8. Redundant restriction in `approve`

Risk Level: Info

Status: Fixed in the commit f734b1.

Contracts:

- contracts/A7A5.sol

Description:

In the `approve()` function, there is a redundant check that ensures `msg.sender` is not the zero address (`address(0)`). This check is unnecessary because `msg.sender` can never be the zero address.

```
function approve(address _spender, uint256 _value) public returns (bool) {
    require(msg.sender != address(0), "can't approve by zero address"); //
@audit redundant
    _allowances[msg.sender][_spender] = _value;
    emit Approval(msg.sender, _spender, _value);
    return true;
}
```

Remediation:

Consider removing the redundant `require` statement.

5.9. Typo in the revert message

Risk Level: Info

Status: Fixed in the commit f734b1.

Contracts:

- contracts/A7A5.sol

Description:

In the `onlyCompliance` modifier, there is a typo in the error message string.

```
modifier onlyCompliance() {
      require(msg.sender == compliance, "not complience"); // @audit typo
      _;
  }
```

Remediation:

Consider correcting the error message.

6. Appendix

6.1. About us

The Decurity team consists of experienced hackers who have been doing application security assessments and penetration testing for over a decade.

During the recent years, we've gained expertise in the blockchain field and have conducted numerous audits for both centralized and decentralized projects: exchanges, protocols, and blockchain nodes.

Our efforts have helped to protect hundreds of millions of dollars and make web3 a safer place.