



LiQ Quotient Protocol Audit Report

Prepared by: [Nemi](#)

Table of Contents

- Introduction
 - Disclaimer
 - Scope of Audit
 - Methodology
 - Security Review Summary
 - Findings Summary
 - Detailed Findings
-

Introduction

The purpose of this report is to document the findings from the security audit of `Vault.sol` and `Strategy1st.sol` contracts. This audit was conducted to identify potential vulnerabilities and provide actionable recommendations to improve the contract's security and adherence to best practices.

Disclaimer

This report is based on the information provided at the time of the audit and does not guarantee the absence of future vulnerabilities. Subsequent security review and on-chain monitoring are strongly recommended.

Scope of Audit

The audit focused on the following aspects:

- Security vulnerabilities
- Code correctness and logic
- Adherence to best practices
- Gas efficiency

Methodology

The audit process involved:

- Manual code review
 - Automated analysis using Slither and Aderyn
 - Scenario-based testing using Foundry
-

Security Review Summary

The security review was carried out from May 21st, 2025 to May 31st, 2025.

review commit hash: `4a3d56800fb5e647ef99c681d522bbdac6b5f225`

Contract Scope The following smart contracts were in scope of the audit:

- `Vault.sol`
- `Strategy1st.sol`

The following number of issues were found, categorized by their severity:

- **High: 1 issue**
- **Medium: 0 issues**
- **Low: 4 issues**

Findings Summary

ID	Title	Severity	Status
H-1	No access control on initialize function in vault contract	High	Fixed
L-1	Incompatible OpenZeppelin Version Usage in Yearn Vaults Integration	Low	Fixed
L-2	Incorrect Token Metadata Concatenation in Vault Initialization	Low	Fixed
L-3	Unsafe String Concatenation Using abi.encodePacked	Low	Fixed
L-4	Use existing OZ ECDSA library for signature recovery	Low	Fixed

Detailed Findings

[H-1] No access control on initialize function in vault contract

Description

The `initialize` function in the `stMNT` Vault contract lacks access control, allowing any external caller to invoke it. This function is intended to be called only once to set up critical parameters such as the token, governance, management, rewards, and guardian addresses, as well as the vault's name, symbol, and fees. Since there is no restriction on who can call this function, an attacker could potentially call it immediately after deployment to initialize the vault with malicious parameters, overriding the intended configuration

Impact

- **High Severity:** An attacker could front-run the legitimate initialization process and set themselves as the governance or management address, effectively taking control of the vault.
- **Loss of Control:** The vault's intended governance could lose control over critical functions like strategy management, fee updates, or emergency shutdown.
- **Financial Risk:** Malicious parameters (e.g., incorrect token address or excessive fees) could lead to loss of funds for depositors or improper operation of the vault.
- **Trust and Functionality:** Users may lose trust in the protocol if the vault is initialized with unintended parameters, potentially disrupting its core functionality.

Recommended Mitigation

Move the initialization logic to the constructor to ensure it's executed atomically during deployment, preventing front-running and unauthorized calls. The constructor is only callable once by the deployer,

inherently restricting access.

[L-1] Incompatible OpenZeppelin Version Usage in Yearn Vaults Integration

Description

The `strategy1st` contract uses OpenZeppelin v5.3.0 for its main contracts while integrating with Yearn Vaults which depends on OpenZeppelin v4.7.1. This version mismatch causes compilation errors due to the removal of the `safeApprove` method in OpenZeppelin v5.3.0. The Yearn Vaults BaseStrategy contract specifically relies on this method for token approvals.

Impact

- Compilation failures when building the project
- Unexpected behaviour/Potential security implications after deployment because v4.7.1 OZ version is used to interact with yearn contracts but entire project is built with v5.3

Recommended Mitigation

1. Install OpenZeppelin v4.7.1

```
forge install openzeppelin-contracts@v4.7.1=OpenZeppelin/openzeppelin-contracts@v4.7.1 --no-commit
```

2. Update the remappings

```
+ @openzeppelin-contract@5.3.0=lib/openzeppelin-contracts/  
+ @openzeppelin-contracts@4.7.1/=lib/openzeppelin-contracts@v4.7.1/  
  
- @openzeppelin/=lib/openzeppelin-contracts/
```

3. Update the imports in Yearn vaults contracts to use version-specific imports

// For v4.7.1 contracts (Yearn vaults) Example `BaseVault` contract where we had original `safeApprove` issue

```
+ import "@openzeppelin/contracts@4.7.1/token/ERC20/IERC20.sol";  
+ import "@openzeppelin/contracts@4.7.1/token/ERC20/utils/SafeERC20.sol";  
  
- import {IERC20} from "@openzeppelin/contracts/token/ERC20/IERC20.sol";  
- import {ERC20} from "@openzeppelin/contracts/token/ERC20/ERC20.sol";  
- import {SafeERC20} from  
"@openzeppelin/contracts/token/ERC20/utils/SafeERC20.sol";
```

// For v5.3.0 contracts (stMNT) In all stMNT contracts that use OZ dependency Example -> **Strategy1st**
Contract

```
+ import {BaseStrategy, StrategyParams} from
"@yearnvaults/contracts/BaseStrategy.sol";

+ import {Address} from "@openzeppelin/contracts@5.3.0/utils/Address.sol";
+ import {IERC20} from "@openzeppelin/contracts@5.3.0/token/ERC20/IERC20.sol";
+ import {SafeERC20} from
"@openzeppelin/contracts@5.3.0/token/ERC20/utils/SafeERC20.sol";

+ import {Ownable} from "@openzeppelin/contracts@5.3.0/access/Ownable.sol";
+ import "@openzeppelin/contracts@5.3.0/token/ERC20/utils/SafeERC20.sol";

- import {Address} from "@openzeppelin/contracts@5.3.0/utils/Address.sol";
- import {IERC20} from "@openzeppelin/contracts@5.3.0/token/ERC20/IERC20.sol";
- import {SafeERC20} from
"@openzeppelin/contracts@5.3.0/token/ERC20/utils/SafeERC20.sol";

- import {Ownable} from "@openzeppelin/contracts@5.3.0/access/Ownable.sol";
```

This issue is rated as Low severity because

- It's a development-time issue that prevents compilation
- It has a clear mitigation path

[L-2] Incorrect Token Metadata Concatenation in Vault Initialization

Description

In the `initialize()` function, the vault's name is constructed by concatenating "st" with the token's symbol instead of its name. This creates inconsistent and potentially misleading token metadata since typically a token's display name should be derived from the underlying token's name rather than its symbol.

Impact

Cannot be fixed after initialization due to one-time initialization pattern and could lead to Inconsistent token naming standards.

Recommended Mitigation

```
if (bytes(_nameOverride).length == 0) {
-   name = string.concat("st", IDetailedERC20(_token).symbol());
+   name = string.concat("st", IDetailedERC20(_token).name());
} else {
    name = _nameOverride;
}
```

[L-3] Unsafe String Concatenation Using abi.encodePacked

Description

The `initialize()` function uses `abi.encodePacked()` for string concatenation when setting the vault's name and symbol. Using `abi.encodePacked` with dynamic types like strings can lead to hash collisions

Impact

While not exploitable in this context, using `abi.encodePacked` with strings is considered unsafe as it could potentially lead to hash collisions in other scenarios where the concatenated result is used in more security-critical operations

Recommended Mitigation

Use `string.concat()` instead of `abi.encodePacked()` for string concatenation, which is available since Solidity 0.8.12

```
- name = string(  
-     abi.encodePacked("st", IDetailedERC20(_token).name())  
- );  
+ name = string.concat("st", IDetailedERC20(_token).name());  
  
- symbol = string(  
-     abi.encodePacked("st", IDetailedERC20(_token).symbol())  
- );  
+ symbol = string.concat("st", IDetailedERC20(_token).symbol());
```

[L-4] Use existing OZ ECDSA library for signature recovery

Description

The vault contract implements custom ECDSA signature verification which could lead to errors. Instead OpenZeppelin's ECDSA and EIP 712 should be used as it reduces complexity and chances of making errors during type hash, domain separator construction and so on.

Impact

Low severity. While the current implementation works for basic signature verification

- `ecrecover` is being used and its `v` component is not being checked which could lead to two different valid signatures being produced that are not authentic

Recommended Mitigation

- Use OpenZeppelin's EIP 712 and ECDSA contracts for signature verification and construction of message structs

```

+ import {EIP712} from "@openzeppelin/contracts/utils/cryptography/EIP712.sol";
+ import {ECDSA} from "@openzeppelin/contracts/utils/cryptography/ECDSA.sol";

- contract StMNT is IERC20, ReentrancyGuard {
+ contract StMNT is IERC20, ReentrancyGuard, EIP712("StakingContract", "0.4.6") {
//or whatever the API version of the vault is

- bytes32 public constant DOMAIN_TYPE_HASH = keccak256("EIP712Domain(string
name,string version,uint256 chainId,address verifyingContract)");

    function DOMAIN_SEPARATOR() external view returns (bytes32) {
+     return _domainSeparatorV4();
-     return _domainSeparator()
    }

-     bytes32 digest = keccak256(
-         abi.encodePacked(
-             "\x19\x01",
-             _domainSeparator(),
-             keccak256(abi.encode(PERMIT_TYPE_HASH, _owner, _spender, _amount,
nonces[_owner], _expiry))
-         )
-     );

+     bytes32 structHash = keccak256(abi.encode(PERMIT_TYPE_HASH, _owner, _spender,
_amount, nonces[_owner], _expiry));

-     require(ecrecover(digest, _v, _r, _s) == _owner, "Vault invalid signature");

+     bytes32 hash_ = _hashTypedDataV4(structHash);
+     address signer_ = ECDSA.recover(hash_, _v, _r, _s);

+     require(signer_ == _owner, "Vault invalid signature");

    // @audit increment the nonce before the allowance
-     allowance[_owner][_spender] = _amount;
-     nonces[_owner] += 1;

+     nonces[_owner] += 1;
+     allowance[_owner][_spender] = _amount;

```