Pre-audit Overview for Takaturn V2

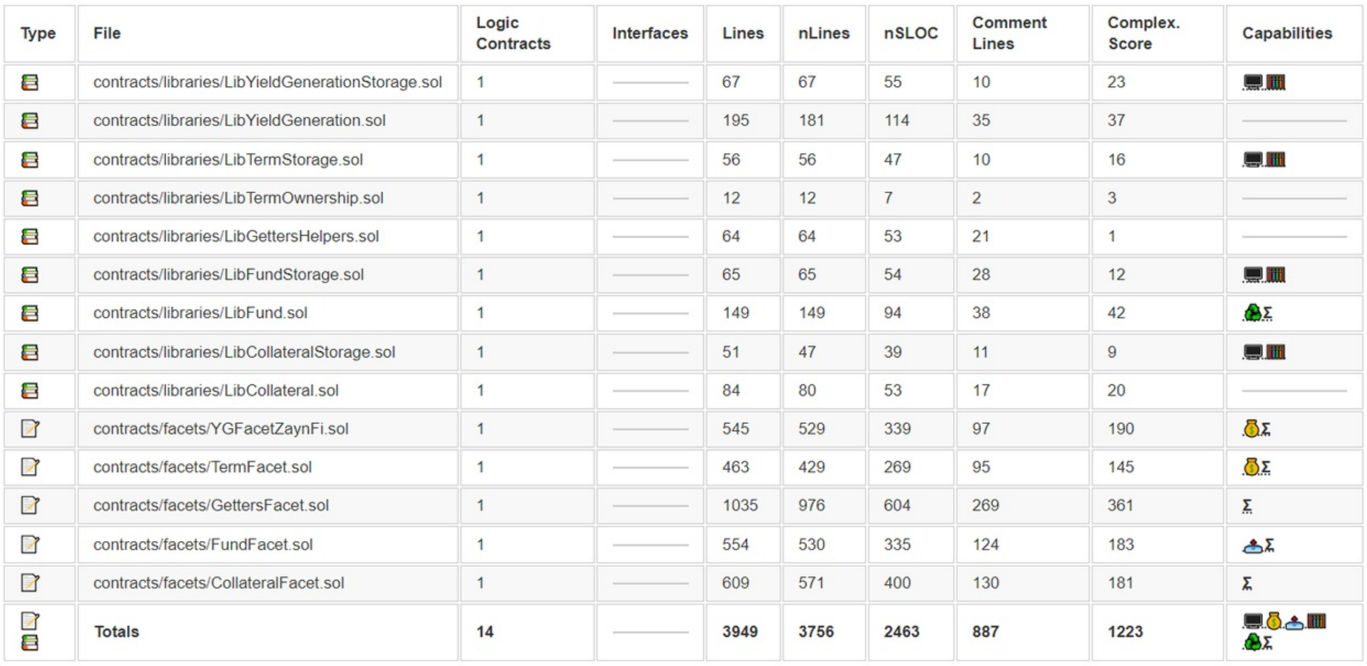
Table of Contents

Introduction

Egis Security’s foremost principle is delivering high-quality work and exceptional service to our clients. That’s why we provide a pre-audit overview with basic issues uncovered during the initial code review.

Scope of the Pre-Audit

The pre-audit was conducted to identify immediate issues that could pose risks to your system. This includes a high-level assessment of:



Type	File	Logic Contracts	Interfaces	Lines	nLines	nSLOC	Comment Lines	Complex. Score	Capabilities
📚	contracts/libraries/LibYieldGenerationStorage.sol	1	----	67	67	55	10	23	🖥🧮
📚	contracts/libraries/LibYieldGeneration.sol	1	----	195	181	114	35	37	----
📚	contracts/libraries/LibTermStorage.sol	1	----	56	56	47	10	16	🖥🧮
📚	contracts/libraries/LibTermOwnership.sol	1	----	12	12	7	2	3	----
📚	contracts/libraries/LibGettersHelpers.sol	1	----	64	64	53	21	1	----
📚	contracts/libraries/LibFundStorage.sol	1	----	65	65	54	28	12	🖥🧮
📚	contracts/libraries/LibFund.sol	1	----	149	149	94	38	42	💸Σ
📚	contracts/libraries/LibCollateralStorage.sol	1	----	51	47	39	11	9	🖥🧮
📚	contracts/libraries/LibCollateral.sol	1	----	84	80	53	17	20	----
📝	contracts/facets/YGFacetZaynFi.sol	1	----	545	529	339	97	190	💰Σ
📝	contracts/facets/TermFacet.sol	1	----	463	429	269	95	145	💰Σ
📝	contracts/facets/GettersFacet.sol	1	----	1035	976	604	269	361	Σ
📝	contracts/facets/FundFacet.sol	1	----	554	530	335	124	183	📤Σ
📝	contracts/facets/CollateralFacet.sol	1	----	609	571	400	130	181	Σ
📝📚	**Totals**	**14**	----	**3949**	**3756**	**2463**	**887**	**1223**	🖥💰📤🧮💸Σ

Preliminary Findings

High 1: Any user can brick starting of a term if he deposits on last position with min amount

Description
When a new fund is created, system require all participants to be overcollaterized. (have deposited ether, which is valued to at least `allCycles * contributionAmount`). The problem is that when a user joins a term, he is not enforced to provide this value. Instead, he can provide `150% of contributionAmount` if he calls `joinTermOnPosition` with last index:

```
    function minCollateralToDeposit(
        uint termId,
        uint depositorIndex
    ) public view returns (uint amount) {
        LibTermStorage.Term storage term = LibTermStorage._termStorage().terms[termId];

        require(depositorIndex < term.totalParticipants, "TT-GF-01");

        uint contributionAmountInWei = getToCollateralConversionRate(
            term.contributionAmount * 10 ** 18
        );

        amount = (contributionAmountInWei * (term.totalParticipants - depositorIndex) * 150) / 100;
    }
```

The following will result in reverts when `startTerm` for the corresponding term is called. Furthermore, all honest participants who have opted in with enough collateral will have their funds freezed. If `term.totalParticipants` is reached, users can't withdraw collateral, because the state is still `AcceptingCollateral`.

Recommendation
Consider:

- enforcing user to provide all collateral when he joins a term or
- Create a function for user to leave a term, if it hasn't started

Medium 1: Malicious party can call `paySecurityOnBehalfOf` with `optYield = false` for a victim

Description In `TermFacet` there is a function `paySecurityOnBehalfOf`, which enroll provided address for a given term and passing corresponding params. The problem is that a victim may want to join a term with `optYield = true`, but a malicious party can front-run him and call it with `optYield = false`. If the registration period has passed and enough participants has joined, the attacker can combine the tx with a call to `startTerm`. The following will make it impossible for the victim to call `YGFacetZaynFi#toggleOptInYG`, because collateral state will be set to `CycleOngoing`.

Recommendation Modify `paySecurityOnBehalfOf` function to only increase collateral for other users, if they have joined corresponding term.

Low 1: Protocol assumes `stableToken` will always have 6 decimals

Description Protocol assumes that token used in terms will always have 6 decimals, because it is a stable token.

```
  function _payContribution(
        uint _termId,
        address _payer,
        address _participant,
        bool _payNextCycle
    ) internal {
        LibFundStorage.Fund storage fund = LibFundStorage._fundStorage().funds[_termId];
        LibTermStorage.Term storage term = LibTermStorage._termStorage().terms[_termId];

        // Get the amount and do the actual transfer
        // This will only succeed if the sender approved this contract address beforehand
        uint amount = term.contributionAmount * 10 ** 6; // Deducted from user's wallet, six decimals

        bool success = fund.stableToken.transferFrom(_payer, address(this), amount);
    }
```

But DAI is an example of a stable token with 18 decimals. While the following may not be an issue directly, term creator should consider that the amount provided for `contributionAmount` should be scaled correspondingly, which may lead to unexpected problems later. **Recommendation** Consider scaling `contributionAmount` by `IERC20(stableTokenAddress).decimals()`, instead of `10 ** 6`.