// HALBORN

TENET - LLSD Contracts

Smart Contract Security Assessment

Prepared by: **Halborn**

Date of Engagement: **July 3rd, 2023 - July 5th, 2023**

Visit: **Halborn.com**

DOCUMENT REVISION HISTORY

VERSION	MODIFICATION	DATE
0.1	Document Creation	07/07/2023
0.2	Document Updates	07/07/2023
0.3	Draft Review	07/10/2023
0.4	Draft Review	07/10/2023
0.5	Document Updates	07/17/2023
1.0	Remediation Plan	07/17/2023
1.1	Remediation Plan Review	07/19/2023

CONTACTS

CONTACT	COMPANY	EMAIL
Rob Behnke	Halborn	Rob.Behnke@halborn.com
Steven Walbroehl	Halborn	Steven.Walbroehl@halborn.com
Gabi Urrutia	Halborn	Gabi.Urrutia@halborn.com

EXECUTIVE OVERVIEW

1.1 INTRODUCTION

Tenet is a PoS blockchain that innovatively allows not only its native token TENET to maintain network security, but also allows the Omni-Chain tLSD token to protect the network. The LLSD contracts allow users to participate in the PoS and earn rewards proportionally to their stake by delegating their stake to the validators.

TENET engaged Halborn to conduct a security assessment on their LLSD smart contracts beginning on July 3rd, 2023 and ending on July 5th, 2023. The security assessment was scoped to the smart contracts provided to the Halborn team.

1.2 ASSESSMENT SUMMARY

The team at Halborn was provided 3 days for the engagement and assigned a full-time security engineer to assessment the security of the smart contract. The security engineer is a blockchain and smart-contract security expert with advanced penetration testing, smart-contract hacking, and deep knowledge of multiple blockchain protocols.

The purpose of this assessment is to:

- Ensure that smart contract functions operate as intended.
- Identify potential security issues with the smart contracts.

In summary, Halborn did not identify any security risk.

1.3 SCOPE

1. IN-SCOPE:

The security assessment was scoped to the following smart contracts:

- contracts/Factory.sol.
- contracts/Interfaces.sol.
- contracts/TLSD.sol.

Commit ID: 41d438e533a1af4a00435c01ab0066a01cf690c4

1.4 TEST APPROACH & METHODOLOGY

Halborn performed a combination of manual and automated security testing to balance efficiency, timeliness, practicality, and accuracy in regard to the scope of this assessment. While manual testing is recommended to uncover flaws in logic, process, and implementation; automated testing techniques help enhance coverage of the bridge code and can quickly identify items that do not follow security best practices. The following phases and associated tools were used throughout the term of the assessment:

- Research into architecture and purpose.
- Smart contract manual code review and walkthrough.
- Graphing out functionality and contract logic/connectivity/functions. (solgraph)
- Manual assessment of use and safety for the critical Solidity variables and functions in scope to identify any arithmetic related vulnerability classes.
- Manual testing by custom scripts.
- Scanning of solidity files for vulnerabilities, security hotspots or bugs. (MythX)
- Static Analysis of security for scoped contract, and imported functions. (Slither)
- Testnet deployment. (Foundry)

2. RISK METHODOLOGY

Every vulnerability and issue observed by Halborn is ranked based on **two sets** of **Metrics** and a **Severity Coefficient**. This system is inspired by the industry standard Common Vulnerability Scoring System.

The two **Metric sets** are: **Exploitability** and **Impact**. **Exploitability** captures the ease and technical means by which vulnerabilities can be exploited and **Impact** describes the consequences of a successful exploit.

The **Severity Coefficients** is designed to further refine the accuracy of the ranking with two factors: **Reversibility** and **Scope**. These capture the impact of the vulnerability on the environment as well as the number of users and smart contracts affected.

The final score is a value between 0-10 rounded up to 1 decimal place and 10 corresponding to the highest security risk. This provides an objective and accurate rating of the severity of security vulnerabilities in smart contracts.

The system is designed to assist in identifying and prioritizing vulnerabilities based on their level of risk to address the most critical issues in a timely manner.

2.1 EXPLOITABILITY

Attack Origin (AO):

Captures whether the attack requires compromising a specific account.

Attack Cost (AC):

Captures the cost of exploiting the vulnerability incurred by the attacker relative to sending a single transaction on the relevant blockchain. Includes but is not limited to financial and computational cost.

Attack Complexity (AX):

Describes the conditions beyond the attacker's control that must exist in order to exploit the vulnerability. Includes but is not limited to macro situation, available third-party liquidity and regulatory challenges.

Metrics:

Exploitability Metric (m_E)	Metric Value	Numerical Value
Attack Origin (AO)	Arbitrary (AO:A) Specific (AO:S)	1 0.2
Attack Cost (AC)	Low (AC:L) Medium (AC:M) High (AC:H)	1 0.67 0.33
Attack Complexity (AX)	Low (AX:L) Medium (AX:M) High (AX:H)	1 0.67 0.33

Exploitability E is calculated using the following formula:

$$E = \prod m_e$$

2.2 IMPACT

Confidentiality (C):

Measures the impact to the confidentiality of the information resources managed by the contract due to a successfully exploited vulnerability. Confidentiality refers to limiting access to authorized users only.

Integrity (I):

Measures the impact to integrity of a successfully exploited vulnerability. Integrity refers to the trustworthiness and veracity of data stored and/or processed on-chain. Integrity impact directly affecting Deposit or Yield records is excluded.

Availability (A):

Measures the impact to the availability of the impacted component resulting from a successfully exploited vulnerability. This metric refers to smart contract features and functionality, not state. Availability impact directly affecting Deposit or Yield is excluded.

Deposit (D):

Measures the impact to the deposits made to the contract by either users or owners.

Yield (Y):

Measures the impact to the yield generated by the contract for either users or owners.

Metrics:

Impact Metric (m_I)	Metric Value	Numerical Value
Confidentiality (C)	None (I:N)	0
	Low (I:L)	0.25
	Medium (I:M)	0.5
	High (I:H)	0.75
	Critical (I:C)	1
Integrity (I)	None (I:N)	0
	Low (I:L)	0.25
	Medium (I:M)	0.5
	High (I:H)	0.75
	Critical (I:C)	1
Availability (A)	None (A:N)	0
	Low (A:L)	0.25
	Medium (A:M)	0.5
	High (A:H)	0.75
	Critical	1
Deposit (D)	None (D:N)	0
	Low (D:L)	0.25
	Medium (D:M)	0.5
	High (D:H)	0.75
	Critical (D:C)	1
Yield (Y)	None (Y:N)	0
	Low (Y:L)	0.25
	Medium: (Y:M)	0.5
	High: (Y:H)	0.75
	Critical (Y:H)	1

Impact I is calculated using the following formula:

$$I = max(m_I) + \frac{\sum m_I - max(m_I)}{4}$$

2.3 SEVERITY COEFFICIENT

Reversibility (R):

Describes the share of the exploited vulnerability effects that can be reversed. For upgradeable contracts, assume the contract private key is available.

Scope (S):

Captures whether a vulnerability in one vulnerable contract impacts resources in other contracts.

Coefficient (C)	Coefficient Value	Numerical Value
Reversibility (r)	None (R:N) Partial (R:P) Full (R:F)	1 0.5 0.25
Scope (s)	Changed (S:C) Unchanged (S:U)	1.25 1

Severity Coefficient C is obtained by the following product:

$$C = rs$$

The Vulnerability Severity Score S is obtained by:

$$S = min(10, EIC * 10)$$

The score is rounded up to 1 decimal places.

Severity	Score Value Range
Critical	9 - 10
High	7 - 8.9
Medium	4.5 - 6.9
Low	2 - 4.4
Informational	0 - 1.9

3. ASSESSMENT SUMMARY & FINDINGS OVERVIEW

CRITICAL	HIGH	MEDIUM	LOW	INFORMATIONAL
0	0	0	0	0

MANUAL TESTING

MANUAL TESTING

The main goal of the manual testing performed during this assessment was to test all the functionalities regarding the TENET LLSD contracts, focusing on the following points/scenarios:

1. Testing the critical and basic protocol functionalities (delegate native and non-native tokens, undelegating native and non-native tokens, increasing, and decreasing delegations).
2. Tests focused on user's delegation to validators (holder of TENET tokens and holder of LSD tokens participating in the Diversified PoS consensus mechanism and earning a portion of staking rewards).
3. Tests focused on user's undelegation from validators (holders of TLSD tokens undelegating completely or a portion of their stake from the network).
4. Tests focused on claiming validators rewards (claiming delegation rewards earned by a specific validator using a TLSD token).
5. Tests focused on the creation of new TLSD tokens within the contract (issuing new TLSD tokens from the TLSDFactory contract by delegating previously whitelisted assets to the PoS mechanism).
6. Tests focused on the difference between issue and issueNative functionalities (using the different Cosmos SDK modules, alliance and staking modules, and analyzing the differences between them at the precompile level).

MANUAL TESTING

Test	Result
Issue tLSD tokens using a valid asset for delegation	Pass
Not possible to issue tLSD tokens for an invalid asset not whitelisted by governance	Pass
Not possible for the user to undelegate a greater portion than previously obtained while delegation	Pass
Issue tLSD tokens using the native token for delegation	Pass
Not possible to reach a DoS when users burning their tLSD tokens previously obtained by delegation	Pass
Users are able to change their stake from one validator to another using the same valid asset for delegation	Pass
Validators are able to claim their earned delegation rewards	Pass
The storage of assets and validators are properly stored and no collision is possible	Pass
The creation of new tLSD tokens within the contract is properly done	Pass
Not possible to exist the same validator for the same asset delegated more than once	Pass
Not possible to reach storage collision when assigning new validator IDs for new validators within the contract	Pass
The conversion from tLSD token to its validator is properly done	Pass
The conversion from tLSD token to its asset is properly done	Pass
Not possible for users to get tLSD tokens without transferring the correct amount of the valid asset linked to that tLSD token	Pass
Only the factory contract can mint and burn tLSD tokens	Pass

AUTOMATED TESTING

5.1 STATIC ANALYSIS REPORT

Description:

Halborn used automated testing techniques to enhance the coverage of certain areas of the scoped contracts. Among the tools used was Slither, a Solidity static analysis framework. After Halborn verified all the contracts in the repository and was able to compile them correctly into their ABI and binary formats, Slither was run on the all-scoped contracts. This tool can statically verify mathematical relationships between Solidity variables to detect invalid or inconsistent usage of the contracts' APIs across the entire code-base.

Slither results:

contracts/Factory.sol

```
MathUpgradeable.mulDiv(uint256,uint256,uint256) (lib/openzeppelin-contracts/contracts-upgradeable/utils/math/MathUpgradeable.sol#55-135) performs a multiplication on the result of a division:
        -denominator = denominator / twos (lib/openzeppelin-contracts/contracts-upgradeable/utils/math/MathUpgradeable.sol#102)
        -inverse = (3 * denominator) ^ 2 (lib/openzeppelin-contracts/contracts-upgradeable/utils/math/MathUpgradeable.sol#117)
MathUpgradeable.mulDiv(uint256,uint256,uint256) (lib/openzeppelin-contracts/contracts-upgradeable/utils/math/MathUpgradeable.sol#55-135) performs a multiplication on the result of a division:
        -denominator = denominator / twos (lib/openzeppelin-contracts/contracts-upgradeable/utils/math/MathUpgradeable.sol#102)
        -inverse += 2 - denominator * inverse (lib/openzeppelin-contracts/contracts-upgradeable/utils/math/MathUpgradeable.sol#121)
MathUpgradeable.mulDiv(uint256,uint256,uint256) (lib/openzeppelin-contracts/contracts-upgradeable/utils/math/MathUpgradeable.sol#55-135) performs a multiplication on the result of a division:
        -denominator = denominator / twos (lib/openzeppelin-contracts/contracts-upgradeable/utils/math/MathUpgradeable.sol#102)
        -inverse += 2 - denominator * inverse (lib/openzeppelin-contracts/contracts-upgradeable/utils/math/MathUpgradeable.sol#122)
MathUpgradeable.mulDiv(uint256,uint256,uint256) (lib/openzeppelin-contracts/contracts-upgradeable/utils/math/MathUpgradeable.sol#55-135) performs a multiplication on the result of a division:
        -denominator = denominator / twos (lib/openzeppelin-contracts/contracts-upgradeable/utils/math/MathUpgradeable.sol#102)
        -inverse += 2 - denominator * inverse (lib/openzeppelin-contracts/contracts-upgradeable/utils/math/MathUpgradeable.sol#123)
MathUpgradeable.mulDiv(uint256,uint256,uint256) (lib/openzeppelin-contracts/contracts-upgradeable/utils/math/MathUpgradeable.sol#55-135) performs a multiplication on the result of a division:
        -denominator = denominator / twos (lib/openzeppelin-contracts/contracts-upgradeable/utils/math/MathUpgradeable.sol#102)
        -inverse += 2 - denominator * inverse (lib/openzeppelin-contracts/contracts-upgradeable/utils/math/MathUpgradeable.sol#124)
MathUpgradeable.mulDiv(uint256,uint256,uint256) (lib/openzeppelin-contracts/contracts-upgradeable/utils/math/MathUpgradeable.sol#55-135) performs a multiplication on the result of a division:
        -denominator = denominator / twos (lib/openzeppelin-contracts/contracts-upgradeable/utils/math/MathUpgradeable.sol#102)
        -inverse += 2 - denominator * inverse (lib/openzeppelin-contracts/contracts-upgradeable/utils/math/MathUpgradeable.sol#125)
MathUpgradeable.mulDiv(uint256,uint256,uint256) (lib/openzeppelin-contracts/contracts-upgradeable/utils/math/MathUpgradeable.sol#55-135) performs a multiplication on the result of a division:
        -denominator = denominator / twos (lib/openzeppelin-contracts/contracts-upgradeable/utils/math/MathUpgradeable.sol#102)
        -inverse += 2 - denominator * inverse (lib/openzeppelin-contracts/contracts-upgradeable/utils/math/MathUpgradeable.sol#126)
MathUpgradeable.mulDiv(uint256,uint256,uint256) (lib/openzeppelin-contracts/contracts-upgradeable/utils/math/MathUpgradeable.sol#55-135) performs a multiplication on the result of a division:
        -prod0 = prod0 / twos (lib/openzeppelin-contracts/contracts-upgradeable/utils/math/MathUpgradeable.sol#105)
        -result = prod0 * inverse (lib/openzeppelin-contracts/contracts-upgradeable/utils/math/MathUpgradeable.sol#132)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#divide-before-multiply

TLSDFactory.issue(address,address,uint256) (src/Factory.sol#39-61) ignores return value by allianceStaking.delegate(_validator,_asset,_amount) (src/Factory.sol#49)
TLSDFactory.issueNative(address) (src/Factory.sol#63-81) ignores return value by staking.delegate{value: msg.value}(_validator) (src/Factory.sol#69)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#unused-return

TLSD.constructor(string,string)._name (src/TLSD.sol#9) shadows:
        - ERC20._name (lib/openzeppelin-contracts/contracts/token/ERC20/ERC20.sol#42) (state variable)
TLSD.constructor(string,string)._symbol (src/TLSD.sol#9) shadows:
        - ERC20._symbol (lib/openzeppelin-contracts/contracts/token/ERC20/ERC20.sol#43) (state variable)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#local-variable-shadowing

Reentrancy in TLSDFactory.issue(address,address,uint256) (src/Factory.sol#39-61):
        External calls:
        - IERC20Upgradeable(_asset).safeTransferFrom(msg.sender,address(this),_amount) (src/Factory.sol#44)
        State variables written after the call(s):
        - tlsd = getTLSD(_validator,_asset) (src/Factory.sol#46)
                - TLSDs.push(tlsd) (src/Factory.sol#162)
        - tlsd = getTLSD(_validator,_asset) (src/Factory.sol#46)
                - info[address(tlsd)] = TLSDInfo(_validator,_asset) (src/Factory.sol#165)
        - tlsd = getTLSD(_validator,_asset) (src/Factory.sol#46)
                - lastValidatorId += 1 (src/Factory.sol#151)
        - tlsd = getTLSD(_validator,_asset) (src/Factory.sol#46)
                - validatorAssetTLSD[_validator][_asset] = address(tlsd) (src/Factory.sol#164)
        - tlsd = getTLSD(_validator,_asset) (src/Factory.sol#46)
                - validatorIds[_validator] = lastValidatorId (src/Factory.sol#153)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#reentrancy-vulnerabilities-2

AddressUpgradeable._revert(bytes,string) (lib/openzeppelin-contracts/contracts-upgradeable/utils/AddressUpgradeable.sol#206-218) uses assembly
        - INLINE ASM (lib/openzeppelin-contracts/contracts-upgradeable/utils/AddressUpgradeable.sol#211-214)
StringsUpgradeable.toString(uint256) (lib/openzeppelin-contracts/contracts-upgradeable/utils/StringsUpgradeable.sol#18-38) uses assembly
        - INLINE ASM (lib/openzeppelin-contracts/contracts-upgradeable/utils/StringsUpgradeable.sol#24-26)
        - INLINE ASM (lib/openzeppelin-contracts/contracts-upgradeable/utils/StringsUpgradeable.sol#30-32)
MathUpgradeable.mulDiv(uint256,uint256,uint256) (lib/openzeppelin-contracts/contracts-upgradeable/utils/math/MathUpgradeable.sol#55-135) uses assembly
        - INLINE ASM (lib/openzeppelin-contracts/contracts-upgradeable/utils/math/MathUpgradeable.sol#66-70)
        - INLINE ASM (lib/openzeppelin-contracts/contracts-upgradeable/utils/math/MathUpgradeable.sol#86-93)
        - INLINE ASM (lib/openzeppelin-contracts/contracts-upgradeable/utils/math/MathUpgradeable.sol#100-109)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#assembly-usage
```

contracts/Interfaces.sol

```
Pragma version^0.8.0 (src/Interfaces.sol#3) allows old versions
solc-0.8.13 is not recommended for deployment
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#incorrect-versions-of-solidity

Function ITLSDFactory.TLSDToAsset(address,uint256) (src/Interfaces.sol#93) is not in mixedCase
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#conformance-to-solidity-naming-conventions
```

AUTOMATED TESTING

contracts/TLSD.sol

```
TLSD.constructor(string,string)._name (src/TLSD.sol#9) shadows:
        - ERC20._name (lib/openzeppelin-contracts/contracts/token/ERC20/ERC20.sol#42) (state variable)
TLSD.constructor(string,string)._symbol (src/TLSD.sol#9) shadows:
        - ERC20._symbol (lib/openzeppelin-contracts/contracts/token/ERC20/ERC20.sol#43) (state variable)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#local-variable-shadowing

Different versions of Solidity are used:
        - Version used: ['^0.8.0', '^0.8.12']
        - ^0.8.0 (lib/openzeppelin-contracts/contracts/access/Ownable.sol#4)
        - ^0.8.0 (lib/openzeppelin-contracts/contracts/token/ERC20/ERC20.sol#4)
        - ^0.8.0 (lib/openzeppelin-contracts/contracts/token/ERC20/IERC20.sol#4)
        - ^0.8.0 (lib/openzeppelin-contracts/contracts/token/ERC20/extensions/IERC20Metadata.sol#4)
        - ^0.8.0 (lib/openzeppelin-contracts/contracts/utils/Context.sol#4)
        - ^0.8.12 (src/TLSD.sol#3)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#different-pragma-directives-are-used

Context._msgData() (lib/openzeppelin-contracts/contracts/utils/Context.sol#21-23) is never used and should be removed
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#dead-code

Pragma version^0.8.0 (lib/openzeppelin-contracts/contracts/access/Ownable.sol#4) allows old versions
Pragma version^0.8.0 (lib/openzeppelin-contracts/contracts/token/ERC20/ERC20.sol#4) allows old versions
Pragma version^0.8.0 (lib/openzeppelin-contracts/contracts/token/ERC20/IERC20.sol#4) allows old versions
Pragma version^0.8.0 (lib/openzeppelin-contracts/contracts/token/ERC20/extensions/IERC20Metadata.sol#4) allows old versions
Pragma version^0.8.0 (lib/openzeppelin-contracts/contracts/utils/Context.sol#4) allows old versions
Pragma version^0.8.12 (src/TLSD.sol#3) necessitates a version too recent to be trusted. Consider deploying with 0.6.12/0.7.6/0.8.7
solc-0.8.13 is not recommended for deployment
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#incorrect-versions-of-solidity

Parameter TLSD.mint(address,uint256)._to (src/TLSD.sol#11) is not in mixedCase
Parameter TLSD.mint(address,uint256)._amount (src/TLSD.sol#11) is not in mixedCase
Parameter TLSD.burn(address,uint256)._from (src/TLSD.sol#15) is not in mixedCase
Parameter TLSD.burn(address,uint256)._amount (src/TLSD.sol#15) is not in mixedCase
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#conformance-to-solidity-naming-conventions

renounceOwnership() should be declared external:
        - Ownable.renounceOwnership() (lib/openzeppelin-contracts/contracts/access/Ownable.sol#61-63)
transferOwnership(address) should be declared external:
        - Ownable.transferOwnership(address) (lib/openzeppelin-contracts/contracts/access/Ownable.sol#69-72)
name() should be declared external:
        - ERC20.name() (lib/openzeppelin-contracts/contracts/token/ERC20/ERC20.sol#62-64)
symbol() should be declared external:
        - ERC20.symbol() (lib/openzeppelin-contracts/contracts/token/ERC20/ERC20.sol#70-72)
decimals() should be declared external:
        - ERC20.decimals() (lib/openzeppelin-contracts/contracts/token/ERC20/ERC20.sol#87-89)
totalSupply() should be declared external:
        - ERC20.totalSupply() (lib/openzeppelin-contracts/contracts/token/ERC20/ERC20.sol#94-96)
balanceOf(address) should be declared external:
        - ERC20.balanceOf(address) (lib/openzeppelin-contracts/contracts/token/ERC20/ERC20.sol#101-103)
transfer(address,uint256) should be declared external:
        - ERC20.transfer(address,uint256) (lib/openzeppelin-contracts/contracts/token/ERC20/ERC20.sol#113-117)
approve(address,uint256) should be declared external:
        - ERC20.approve(address,uint256) (lib/openzeppelin-contracts/contracts/token/ERC20/ERC20.sol#136-140)
transferFrom(address,address,uint256) should be declared external:
        - ERC20.transferFrom(address,address,uint256) (lib/openzeppelin-contracts/contracts/token/ERC20/ERC20.sol#158-167)
increaseAllowance(address,uint256) should be declared external:
        - ERC20.increaseAllowance(address,uint256) (lib/openzeppelin-contracts/contracts/token/ERC20/ERC20.sol#181-185)
decreaseAllowance(address,uint256) should be declared external:
        - ERC20.decreaseAllowance(address,uint256) (lib/openzeppelin-contracts/contracts/token/ERC20/ERC20.sol#201-210)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#public-function-that-could-be-declared-external
```

- As a result of the tests carried out with the Slither tool, some results were obtained and reviewed by Halborn. Based on the results reviewed, the vulnerabilities were determined to be false positives.

5.2 AUTOMATED SECURITY SCAN

Description:

Halborn used automated security scanners to assist with detection of well-known security issues, and to identify low-hanging fruits on the targets for this engagement. Among the tools used was MythX, a security analysis service for Ethereum smart contracts. MythX performed a scan on all the contracts and sent the compiled results to the analyzers to locate any vulnerabilities.

MythX results:

Report for lib/openzeppelin-contracts/contracts-upgradeable/utils/AddressUpgradeable.sol
https://dashboard.mythx.io/#/console/analyses/4f82bb3a-7e31-469e-be6f-bc4cea628193

Line	SWC Title	Severity	Short Description
135	(SWC-123) Requirement Violation	Low	Requirement violation.

Report for src/Factory.sol
https://dashboard.mythx.io/#/console/analyses/4f82bb3a-7e31-469e-be6f-bc4cea628193

Line	SWC Title	Severity	Short Description
3	(SWC-103) Floating Pragma	Low	A floating pragma is set.
11	(SWC-123) Requirement Violation	Low	Requirement violation.
67	(SWC-113) DoS with Failed Call	Medium	Multiple calls are executed in the same transaction.
86	(SWC-113) DoS with Failed Call	Low	Multiple calls are executed in the same transaction.
86	(SWC-107) Reentrancy	Low	A call to a user-supplied address is executed.
88	(SWC-107) Reentrancy	Medium	Read of persistent state following external call
141	(SWC-107) Reentrancy	Low	Read of persistent state following external call.
146	(SWC-113) DoS with Failed Call	Medium	Multiple calls are executed in the same transaction.
162	(SWC-107) Reentrancy	Low	Read of persistent state following external call.
164	(SWC-107) Reentrancy	Low	Read of persistent state following external call.
165	(SWC-107) Reentrancy	Low	Read of persistent state following external call.
168	(SWC-107) Reentrancy	Low	Read of persistent state following external call.

Report for src/TLSD.sol
https://dashboard.mythx.io/#/console/analyses/1032d400-bd2a-4321-89a7-746b2f5210f9

Line	SWC Title	Severity	Short Description
3	(SWC-103) Floating Pragma	Low	A floating pragma is set.

- No major issues found by Mythx.

THANK YOU FOR CHOOSING

// HALBORN