// HALBORN

Tenderize - Polygon Integration

Smart Contract Security Assessment

Prepared by: **Halborn**

Date of Engagement: **September 19th, 2023 - September 22nd, 2023**

Visit: **Halborn.com**

DOCUMENT REVISION HISTORY

VERSION	MODIFICATION	DATE
0.1	Document Creation	09/20/2023
0.2	Document Updates	09/22/2023
0.3	Draft Version	09/22/2023
0.4	Draft Review	09/25/2023
1.0	Remediation Plan	09/27/2023
1.1	Remediation Plan Review	09/28/2023

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
Gokberk Gulgun	Halborn	Gokberk.Gulgun@halborn.com
Omar Alshaeb	Halborn	Omar.Alshaeb@halborn.com

EXECUTIVE OVERVIEW

1.1 INTRODUCTION

Tenderize engaged Halborn to conduct a security assessment on their smart contracts beginning on September 19th, 2023 and ending on September 22nd, 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 four days for the engagement and assigned a full-time security engineer to verify 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 identified some security risks that were acknowledged and accepted by the Tenderize team.

1.3 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 code and can quickly identify items that do not follow the security best practices. The following phases and associated tools were used during 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 hot-spots 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)	1
	Partial (R:P)	0.5
	Full (R:F)	0.25
Scope (s)	Changed (S:C)	1.25
	Unchanged (S:U)	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

2.4 SCOPE

IN-SCOPE CODE & COMMITS:

- Repository: Tenderize/staking
    - Commit ID: 84d7764d9f8f92f56c7d48b1cd046751eacd818f
    - Smart contracts **in scope**:
        - src/adapters/PolygonAdapter.sol

OUT-OF-SCOPE:

- Third-party libraries and dependencies.
- Economic attacks.

EXECUTIVE OVERVIEW

3. ASSESSMENT SUMMARY & FINDINGS OVERVIEW

CRITICAL	HIGH	MEDIUM	LOW	INFORMATIONAL
0	0	0	2	1

SECURITY ANALYSIS	RISK LEVEL	REMEDIATION DATE
(HAL-01) CONTRACT PAUSE FEATURE MISSING	Low (2.5)	RISK ACCEPTED
(HAL-02) FIXED POINT MATH LIBRARY NOT USED	Low (2.5)	RISK ACCEPTED
(HAL-03) INCOMPLETE NATSPEC DOCUMENTATION	Informational (0.0)	ACKNOWLEDGED

FINDINGS & TECH DETAILS

4.1 (HAL-01) CONTRACT PAUSE FEATURE MISSING - LOW (2.5)

Description:

It was identified that no high-privileged user can pause any of the scoped contracts. In the event of a security incident, the owner would not be able to stop any plausible malicious actions. Pausing the contract can also lead to more considered decisions.

BVSS:

AO:A/AC:L/AX:M/C:N/I:L/A:N/D:L/Y:L/R:N/S:U (2.5)

Recommendation:

Consider adding the pausable functionality to the contract.

Remediation Plan:

RISK ACCEPTED: The Tenderize team accepted the risk of this finding.

4.2 (HAL-02) FIXED POINT MATH LIBRARY NOT USED - LOW (2.5)

Description:

It was identified that FixedPointMathLib library from Solmate is not used when performing mathematical operations within the PolygonAdapter contract. This could lead to rounding issues in the divisions.

Code Location:

Listing 1: src/adapters/PolygonAdapter.sol

```
amount = unbond.shares * validatorShares.withdrawExchangeRate() /
↳ getExchangePrecision(validatorId);
```

Listing 2: src/adapters/PolygonAdapter.sol

```
return u.withdrawEpoch + WITHDRAW_DELAY;
```

Listing 3: src/adapters/PolygonAdapter.sol

```
uint256 min = amount * precision / fxRate - 1;
```

Listing 4: src/adapters/PolygonAdapter.sol

```
uint256 max = amount * precision / fxRate + 1;
```

Listing 5: src/adapters/PolygonAdapter.sol

```
amount = unbond.shares * fxRate / getExchangePrecision(validatorId
↳ );
```

Listing 6: src/adapters/PolygonAdapter.sol

```
newStake = shares * fxRate / precision;
```

BVSS:

AO:A/AC:L/AX:M/C:N/I:L/A:N/D:L/Y:L/R:N/S:U (2.5)

Recommendation:

Consider importing and using the FixedPointMathLib library from Solmate to avoid this type of issue.

Remediation Plan:

RISK ACCEPTED: The Tenderize team accepted the risk of the finding.

4.3 (HAL-03) INCOMPLETE NATSPEC DOCUMENTATION - INFORMATIONAL (0.0)

Description:

Natspec documentation is useful for internal developers that need to work on the project, external developers that need to integrate with the project, security researchers that have to review it but also for end users given that many chain explorers have officially integrated the support for it directly on their site.

It was detected that the scoped contracts have an incomplete **Natspec** documentation.

BVSS:

AO:A/AC:L/AX:L/C:N/I:N/A:N/D:N/Y:N/R:N/S:U (0.0)

Recommendation:

Consider adding the missing **Natspec** documentation, adhering to the format guideline included in Solidity documentation.

Remediation Plan:

ACKNOWLEDGED: The Tenderize team acknowledged this finding.

MANUAL TESTING

The main goal of the manual testing performed during this assessment was to test all the functionalities regarding the PolygonAdapter contract, focusing on the following points:

Test	Result
Matic exchange rate precision used is the correct one depending on validator IDs	Pass
Users cannot withdraw tokens bypassing the withdraw delay, 80 epochs	Pass
Delegation of user's stake is properly delegated to the specified validator ID	Pass
Users can always unstake their tokens as long as they wait for the withdrawal delay to be completed	Pass
No issues related to token approvals between users and contracts	Pass
Not possible to perform a DoS attack by reaching the transaction gas limit	Pass
Not possible to perform a DoS attack by mathematical operations that could overflow	Pass
When creating an unbond request, the data is properly stored in the storage	Pass
When checking an unbond request data, the data is properly checked for the withdrawal of tokens	Pass
When depositing tokens, the amount of shares corresponding to the amount deposited for the specified validator ID cannot be maliciously modified	Pass
Only the owner of the NFT representing the withdrawal request can execute the transaction to get the tokens	Pass

AUTOMATED TESTING

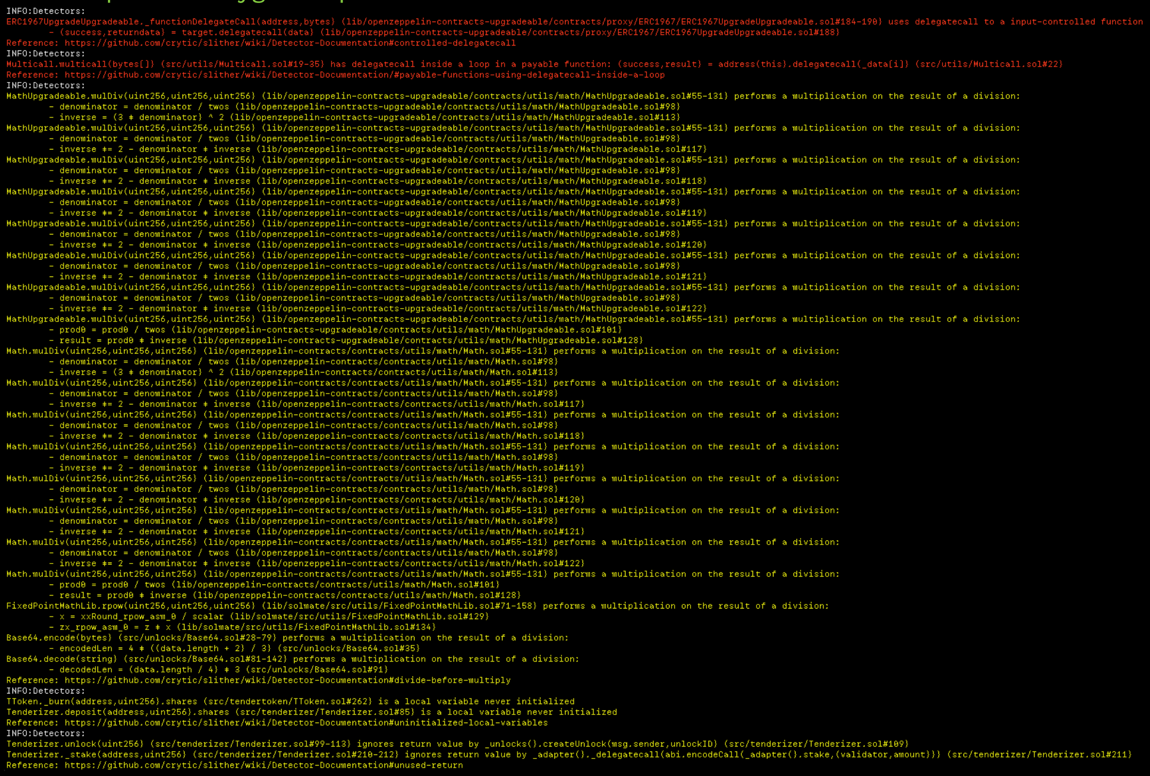
6.1 STATIC ANALYSIS REPORT

Description:

Halborn used automated testing techniques to enhance the coverage of certain areas of the smart contracts in scope. Among the tools used was Slither, a Solidity static analysis framework. After Halborn verified the smart contracts in the repository and was able to compile them correctly into their ABIs and binary format, Slither was run against the 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.

Results:

src/adapters/PolygonAdapter.sol



```
INFO:Detectors:
ERC1967UpgradeUpgradeable._functionDelegateCall(address,bytes) (lib/openzeppelin-contracts-upgradeable/contracts/proxy/ERC1967/ERC1967UpgradeUpgradeable.sol#184-190) uses delegatecall to a input-controlled function
        - (success,returndata) = target.delegatecall(data) (lib/openzeppelin-contracts-upgradeable/contracts/proxy/ERC1967/ERC1967UpgradeUpgradeable.sol#188)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#controlled-delegatecall
INFO:Detectors:
Multicall.multicall(bytes[]) (src/utils/Multicall.sol#19-35) has delegatecall inside a loop in a payable function: (success,result) = address(this).delegatecall(_data[i]) (src/utils/Multicall.sol#22)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation/#payable-functions-using-delegatecall-inside-a-loop
INFO:Detectors:
MathUpgradeable.mulDiv(uint256,uint256,uint256) (lib/openzeppelin-contracts-upgradeable/contracts/utils/math/MathUpgradeable.sol#55-131) performs a multiplication on the result of a division:
        - denominator = denominator / twos (lib/openzeppelin-contracts-upgradeable/contracts/utils/math/MathUpgradeable.sol#98)
        - inverse = (3 * denominator) ^ 2 (lib/openzeppelin-contracts-upgradeable/contracts/utils/math/MathUpgradeable.sol#113)
MathUpgradeable.mulDiv(uint256,uint256,uint256) (lib/openzeppelin-contracts-upgradeable/contracts/utils/math/MathUpgradeable.sol#55-131) performs a multiplication on the result of a division:
        - denominator = denominator / twos (lib/openzeppelin-contracts-upgradeable/contracts/utils/math/MathUpgradeable.sol#98)
        - inverse *= 2 - denominator * inverse (lib/openzeppelin-contracts-upgradeable/contracts/utils/math/MathUpgradeable.sol#117)
MathUpgradeable.mulDiv(uint256,uint256,uint256) (lib/openzeppelin-contracts-upgradeable/contracts/utils/math/MathUpgradeable.sol#55-131) performs a multiplication on the result of a division:
        - denominator = denominator / twos (lib/openzeppelin-contracts-upgradeable/contracts/utils/math/MathUpgradeable.sol#98)
        - inverse *= 2 - denominator * inverse (lib/openzeppelin-contracts-upgradeable/contracts/utils/math/MathUpgradeable.sol#118)
MathUpgradeable.mulDiv(uint256,uint256,uint256) (lib/openzeppelin-contracts-upgradeable/contracts/utils/math/MathUpgradeable.sol#55-131) performs a multiplication on the result of a division:
        - denominator = denominator / twos (lib/openzeppelin-contracts-upgradeable/contracts/utils/math/MathUpgradeable.sol#98)
        - inverse *= 2 - denominator * inverse (lib/openzeppelin-contracts-upgradeable/contracts/utils/math/MathUpgradeable.sol#119)
MathUpgradeable.mulDiv(uint256,uint256,uint256) (lib/openzeppelin-contracts-upgradeable/contracts/utils/math/MathUpgradeable.sol#55-131) performs a multiplication on the result of a division:
        - denominator = denominator / twos (lib/openzeppelin-contracts-upgradeable/contracts/utils/math/MathUpgradeable.sol#98)
        - inverse *= 2 - denominator * inverse (lib/openzeppelin-contracts-upgradeable/contracts/utils/math/MathUpgradeable.sol#120)
MathUpgradeable.mulDiv(uint256,uint256,uint256) (lib/openzeppelin-contracts-upgradeable/contracts/utils/math/MathUpgradeable.sol#55-131) performs a multiplication on the result of a division:
        - denominator = denominator / twos (lib/openzeppelin-contracts-upgradeable/contracts/utils/math/MathUpgradeable.sol#98)
        - inverse *= 2 - denominator * inverse (lib/openzeppelin-contracts-upgradeable/contracts/utils/math/MathUpgradeable.sol#121)
MathUpgradeable.mulDiv(uint256,uint256,uint256) (lib/openzeppelin-contracts-upgradeable/contracts/utils/math/MathUpgradeable.sol#55-131) performs a multiplication on the result of a division:
        - denominator = denominator / twos (lib/openzeppelin-contracts-upgradeable/contracts/utils/math/MathUpgradeable.sol#98)
        - inverse *= 2 - denominator * inverse (lib/openzeppelin-contracts-upgradeable/contracts/utils/math/MathUpgradeable.sol#122)
MathUpgradeable.mulDiv(uint256,uint256,uint256) (lib/openzeppelin-contracts-upgradeable/contracts/utils/math/MathUpgradeable.sol#55-131) performs a multiplication on the result of a division:
        - prod0 = prod0 / twos (lib/openzeppelin-contracts-upgradeable/contracts/utils/math/MathUpgradeable.sol#101)
        - result = prod0 * inverse (lib/openzeppelin-contracts-upgradeable/contracts/utils/math/MathUpgradeable.sol#128)
Math.mulDiv(uint256,uint256,uint256) (lib/openzeppelin-contracts/contracts/utils/math/Math.sol#55-131) performs a multiplication on the result of a division:
        - denominator = denominator / twos (lib/openzeppelin-contracts/contracts/utils/math/Math.sol#98)
        - inverse = (3 * denominator) ^ 2 (lib/openzeppelin-contracts/contracts/utils/math/Math.sol#113)
Math.mulDiv(uint256,uint256,uint256) (lib/openzeppelin-contracts/contracts/utils/math/Math.sol#55-131) performs a multiplication on the result of a division:
        - denominator = denominator / twos (lib/openzeppelin-contracts/contracts/utils/math/Math.sol#98)
        - inverse *= 2 - denominator * inverse (lib/openzeppelin-contracts/contracts/utils/math/Math.sol#117)
Math.mulDiv(uint256,uint256,uint256) (lib/openzeppelin-contracts/contracts/utils/math/Math.sol#55-131) performs a multiplication on the result of a division:
        - denominator = denominator / twos (lib/openzeppelin-contracts/contracts/utils/math/Math.sol#98)
        - inverse *= 2 - denominator * inverse (lib/openzeppelin-contracts/contracts/utils/math/Math.sol#118)
Math.mulDiv(uint256,uint256,uint256) (lib/openzeppelin-contracts/contracts/utils/math/Math.sol#55-131) performs a multiplication on the result of a division:
        - denominator = denominator / twos (lib/openzeppelin-contracts/contracts/utils/math/Math.sol#98)
        - inverse *= 2 - denominator * inverse (lib/openzeppelin-contracts/contracts/utils/math/Math.sol#119)
Math.mulDiv(uint256,uint256,uint256) (lib/openzeppelin-contracts/contracts/utils/math/Math.sol#55-131) performs a multiplication on the result of a division:
        - denominator = denominator / twos (lib/openzeppelin-contracts/contracts/utils/math/Math.sol#98)
        - inverse *= 2 - denominator * inverse (lib/openzeppelin-contracts/contracts/utils/math/Math.sol#120)
Math.mulDiv(uint256,uint256,uint256) (lib/openzeppelin-contracts/contracts/utils/math/Math.sol#55-131) performs a multiplication on the result of a division:
        - denominator = denominator / twos (lib/openzeppelin-contracts/contracts/utils/math/Math.sol#98)
        - inverse *= 2 - denominator * inverse (lib/openzeppelin-contracts/contracts/utils/math/Math.sol#121)
Math.mulDiv(uint256,uint256,uint256) (lib/openzeppelin-contracts/contracts/utils/math/Math.sol#55-131) performs a multiplication on the result of a division:
        - denominator = denominator / twos (lib/openzeppelin-contracts/contracts/utils/math/Math.sol#98)
        - inverse *= 2 - denominator * inverse (lib/openzeppelin-contracts/contracts/utils/math/Math.sol#122)
Math.mulDiv(uint256,uint256,uint256) (lib/openzeppelin-contracts/contracts/utils/math/Math.sol#55-131) performs a multiplication on the result of a division:
        - prod0 = prod0 / twos (lib/openzeppelin-contracts/contracts/utils/math/Math.sol#101)
        - result = prod0 * inverse (lib/openzeppelin-contracts/contracts/utils/math/Math.sol#128)
FixedPointMathLib.rpow(uint256,uint256,uint256) (lib/solmate/src/utils/FixedPointMathLib.sol#71-158) performs a multiplication on the result of a division:
        - x = xxRound_rpow_asm_0 / scalar (lib/solmate/src/utils/FixedPointMathLib.sol#129)
        - zx_rpow_asm_0 = z * x (lib/solmate/src/utils/FixedPointMathLib.sol#134)
Base64.encode(bytes) (src/unlocks/Base64.sol#28-79) performs a multiplication on the result of a division:
        - encodedLen = 4 * ((data.length + 2) / 3) (src/unlocks/Base64.sol#35)
Base64.decode(string) (src/unlocks/Base64.sol#81-142) performs a multiplication on the result of a division:
        - decodedLen = (data.length / 4) * 3 (src/unlocks/Base64.sol#91)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#divide-before-multiply
INFO:Detectors:
TToken._burn(address,uint256).shares (src/tendertoken/TToken.sol#262) is a local variable never initialized
Tenderizer.deposit(address,uint256).shares (src/tenderizer/Tenderizer.sol#85) is a local variable never initialized
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#uninitialized-local-variables
INFO:Detectors:
Tenderizer.unlock(uint256) (src/tenderizer/Tenderizer.sol#99-113) ignores return value by _unlocks().createUnlock(msg.sender,unlockID) (src/tenderizer/Tenderizer.sol#109)
Tenderizer._stake(address,uint256) (src/tenderizer/Tenderizer.sol#210-212) ignores return value by _adapter()._delegatecall(abi.encodeCall(_adapter().stake,(validator,amount))) (src/tenderizer/Tenderizer.sol#211)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#unused-return
```

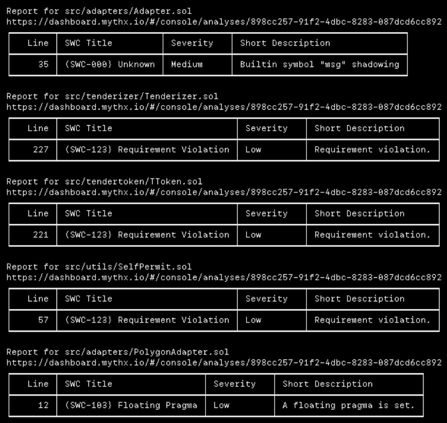
All the issues flagged by Slither were manually reviewed by Halborn. Reported issues were either considered as false positives or are already included in the report findings.

6.2 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 the smart contracts and sent the compiled results to the analyzers in order to locate any vulnerabilities.

Results:



```
Report for src/adapters/Adapter.sol
https://dashboard.mythx.io/#/console/analyses/898cc257-91f2-4dbc-8283-087dcd6cc892
|   Line | SWC Title           | Severity | Short Description              |
|     35 | (SWC-000) Unknown   | Medium   | Builtin symbol "msg" shadowing |

Report for src/tenderizer/Tenderizer.sol
https://dashboard.mythx.io/#/console/analyses/898cc257-91f2-4dbc-8283-087dcd6cc892
|   Line | SWC Title                         | Severity | Short Description      |
|    227 | (SWC-123) Requirement Violation   | Low      | Requirement violation. |

Report for src/tendertoken/TToken.sol
https://dashboard.mythx.io/#/console/analyses/898cc257-91f2-4dbc-8283-087dcd6cc892
|   Line | SWC Title                         | Severity | Short Description      |
|    221 | (SWC-123) Requirement Violation   | Low      | Requirement violation. |

Report for src/utils/SelfPermit.sol
https://dashboard.mythx.io/#/console/analyses/898cc257-91f2-4dbc-8283-087dcd6cc892
|   Line | SWC Title                         | Severity | Short Description      |
|     57 | (SWC-123) Requirement Violation   | Low      | Requirement violation. |

Report for src/adapters/PolygonAdapter.sol
https://dashboard.mythx.io/#/console/analyses/898cc257-91f2-4dbc-8283-087dcd6cc892
|   Line | SWC Title                  | Severity | Short Description         |
|     12 | (SWC-103) Floating Pragma  | Low      | A floating pragma is set. |
```

All the issues flagged by MythX were manually reviewed by Halborn. Reported issues were either considered as false positives or are already included in the report findings.

THANK YOU FOR CHOOSING

HALBORN