

Ten Protocol

Obscu/TEN

HALBORN

Ten Protocol - Obscu/TEN

Prepared by: **HALBORN**

Last Updated 06/18/2025

Date of Engagement: May 5th, 2025 - May 8th, 2025

Summary

100% ⓘ OF ALL REPORTED FINDINGS HAVE BEEN ADDRESSED

ALL FINDINGS	CRITICAL	HIGH	MEDIUM	LOW	INFORMATIONAL
22	0	0	0	7	15

TABLE OF CONTENTS

1. Introduction
2. Assessment summary
3. Test approach and methodology
4. Risk methodology
5. Scope
6. Assessment summary & findings overview
7. Findings & Tech Details
7.1 Missing validation of callbackid existence in canreattemptcallback modifier
7.2 Lack of callback cancellation and lifecycle controls
7.3 Inadequate callback array management
7.4 Single step ownership transfer process
7.5 Fixed token minting regardless of transaction value
7.6 Lack of transaction processing tracking mechanism
7.7 Missing error handling for callbacks
7.8 Unnecessary external call in _internalrefund when refund amount is zero
7.9 Missing input validation
7.10 Use of low-level transfer method
7.11 Floating pragma
7.12 Use of revert strings instead of custom errors
7.13 Unhandled return value in _payforcallback function
7.14 Missing events

7.15 Open to-dos

7.16 Redundant default value assignment

7.17 Public functions can be marked external

7.18 Commented functionality

7.19 Missing variable visibility

7.20 Lack of named mappings

7.21 Unused import

7.22 Unoptimized for loops

8. Automated Testing

1. Introduction

Ten Protocol engaged **Halborn** to conduct a security assessment on their smart contracts beginning on May 5th, 2025 and ending on May 8th, 2025. The security assessment was scoped to the smart contracts provided to Halborn. Commit hashes and further details can be found in the Scope section of this report.

The **Ten Protocol** codebase in scope mainly consists of an infrastructure with an upgradeable token system, cross-chain messaging, and transaction processing mechanisms.

2. Assessment Summary

Halborn was provided 4 days for the engagement and assigned 1 full-time security engineer to review the security of the smart contracts in scope. The engineer is a blockchain and smart contract security expert with advanced penetration testing and smart contract hacking skills, and deep knowledge of multiple blockchain protocols.

The purpose of the assessment is to:

- Identify potential security issues within the smart contracts.
- Ensure that smart contract functionality operates as intended.

In summary, **Halborn** identified some improvements to reduce the likelihood and impact of risks, which were partially addressed by the **Ten Protocol team**. The main ones are the following:

- **Add validation in the `canReattemptCallback` modifier to ensure the `callbackId` exists.**
- **Consider adding functionality to allow callback originators to cancel their pending callbacks when they're no longer needed or desired.**
- **Consider implementing a comprehensive callback management system.**

3. Test Approach And Methodology

Halborn performed a combination of manual review of the code 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 smart contracts 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, purpose and use of the platform.
- Smart contract manual code review and walkthrough to identify any logic issue.
- Thorough assessment of safety and usage of critical Solidity variables and functions in scope that could lead to arithmetic related vulnerabilities.
- Local testing with custom scripts (**Foundry**).
- Fork testing against main networks (**Foundry**).
- Static analysis of security for scoped contract, and imported functions (**Slither**).

4. 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.

4.1 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

EXPLOITABILITY METRIC (M_E)	METRIC VALUE	NUMERICAL VALUE
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:

$$E = \prod m_e$$

4.2 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) Low (I:L) Medium (I:M) High (I:H) Critical (I:C)	0 0.25 0.5 0.75 1

Impact Metric (M_I)	Metric Value	Numerical Value
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 (A:C)	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:C)	1

Impact I is calculated using the following formula:

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

4.3 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.

METRICS:

Severity 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

5. SCOPE

REPOSITORY ^

(a) Repository: [go-ten](#)
(b) Assessed Commit ID: [6cae341](#)
(c) Items in scope:

- src/common/Structs.sol
- src/system/contracts/Fees.sol
- src/system/contracts/PublicCallbacks.sol
- src/system/contracts/SystemDeployer.sol
- src/system/contracts/TransactionPostProcessor.sol
- src/system/utils/ZenBase.sol
- src/ten_erc20/TENToken.sol

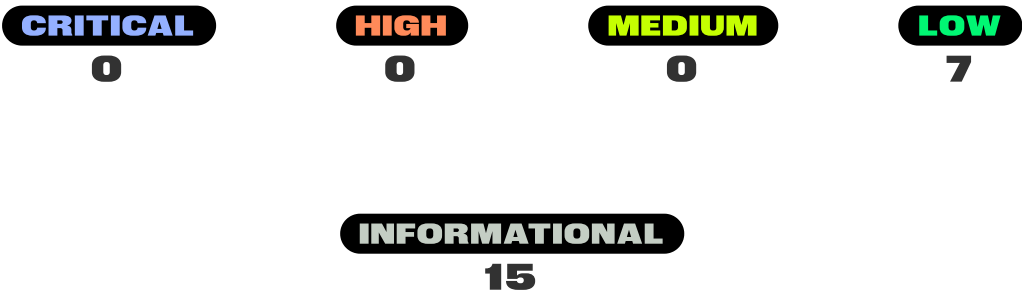
Out-of-Scope: Third party dependencies and economic attacks.

REMEDIATION COMMIT ID: ^

- [fdc1a72](#)

Out-of-Scope: New features/implementations after the remediation commit IDs.

6. ASSESSMENT SUMMARY & FINDINGS OVERVIEW



SECURITY ANALYSIS	RISK LEVEL	REMEDIATION DATE
MISSING VALIDATION OF CALLBACKID EXISTENCE IN CANREATTEMPTCALLBACK MODIFIER	LOW	SOLVED - 05/20/2025
LACK OF CALLBACK CANCELLATION AND LIFECYCLE CONTROLS	LOW	SOLVED - 05/20/2025
INADEQUATE CALLBACK ARRAY MANAGEMENT	LOW	PARTIALLY SOLVED - 05/20/2025
SINGLE STEP OWNERSHIP TRANSFER PROCESS	LOW	PARTIALLY SOLVED - 05/20/2025
FIXED TOKEN MINTING REGARDLESS OF TRANSACTION VALUE	LOW	RISK ACCEPTED - 06/11/2025
LACK OF TRANSACTION PROCESSING TRACKING MECHANISM	LOW	RISK ACCEPTED - 06/11/2025
MISSING ERROR HANDLING FOR CALLBACKS	LOW	RISK ACCEPTED - 06/11/2025
UNNECESSARY EXTERNAL CALL IN _INTERNALREFUND WHEN REFUND AMOUNT IS ZERO	INFORMATIONAL	ACKNOWLEDGED - 06/11/2025
MISSING INPUT VALIDATION	INFORMATIONAL	PARTIALLY SOLVED - 05/20/2025
USE OF LOW-LEVEL TRANSFER METHOD	INFORMATIONAL	SOLVED - 05/20/2025

SECURITY ANALYSIS	RISK LEVEL	REMEDiation DATE
FLOATING PRAGMA	INFORMATIONAL	ACKNOWLEDGED - 06/11/2025
USE OF REVERT STRINGS INSTEAD OF CUSTOM ERRORS	INFORMATIONAL	ACKNOWLEDGED - 06/11/2025
UNHANDLED RETURN VALUE IN _PAYFORCALLBACK FUNCTION	INFORMATIONAL	ACKNOWLEDGED - 06/11/2025
MISSING EVENTS	INFORMATIONAL	SOLVED - 05/20/2025
OPEN TO-DOS	INFORMATIONAL	PARTIALLY SOLVED - 05/20/2025
REDUNDANT DEFAULT VALUE ASSIGNMENT	INFORMATIONAL	SOLVED - 05/20/2025
PUBLIC FUNCTIONS CAN BE MARKED EXTERNAL	INFORMATIONAL	SOLVED - 05/20/2025
COMMENTED FUNCTIONALITY	INFORMATIONAL	PARTIALLY SOLVED - 05/20/2025
MISSING VARIABLE VISIBILITY	INFORMATIONAL	SOLVED - 05/20/2025
LACK OF NAMED MAPPINGS	INFORMATIONAL	SOLVED - 05/20/2025

SECURITY ANALYSIS	RISK LEVEL	REMEDIATION DATE
UNUSED IMPORT	INFORMATIONAL	SOLVED - 05/20/2025
UNOPTIMIZED FOR LOOPS	INFORMATIONAL	SOLVED - 05/20/2025

7. FINDINGS & TECH DETAILS

7.1 MISSING VALIDATION OF CALLBACKID EXISTENCE IN CANREATTEMPTCALLBACK MODIFIER

// LOW

Description

The `canReattemptCallback` modifier in the `PublicCallbacks` contract does not validate whether the provided `callbackId` actually exists before checking its eligibility for reattempt. The modifier only verifies that the callback's block number is less than the current block number, but doesn't check if the callback exists in the first place.

```
946 | modifier canReattemptCallback(uint256 callbackId) {  
947 |     require(callbackBlockNumber[callbackId] < block.number, "Callback cannot be reattempted yet")  
948 |     _;  
949 | }
```

Since these calls succeed rather than revert, users could congest the network with seemingly legitimate but ultimately useless operations. This could artificially inflate transaction volume and potentially disrupt system performance as part of a broader denial-of-service strategy.

BVSS

AO:A/AC:L/AX:L/R:N/S:U/C:N/A:L/I:L/D:N/Y:N (3.1)

Recommendation

Add validation in the `canReattemptCallback` modifier to ensure the callbackId exists.

Remediation Comment

SOLVED: The **Ten Protocol team** solved this finding in commit `fdc1a72` by following the mentioned recommendation.

Remediation Hash

<https://github.com/ten-protocol/go-ten/commit/fdc1a7261852e33e7818c28bdad391e34dbfa7e7>

7.2 LACK OF CALLBACK CANCELLATION AND LIFECYCLE CONTROLS

// LOW

Description

The `PublicCallbacks` contract offers no mechanism for users to cancel registered callbacks that are no longer needed or relevant. Once a callback is registered, it can only be removed from the system through successful execution, and failed callbacks remain permanently in the system with no recourse for users.

This creates a rigid, inflexible system where users have no agency over their registered callbacks after submission, even if circumstances change or the callback is no longer desired.

Additionally, the permissionless nature of `retryCallback()` means anyone can trigger the execution of another user's callback, removing the callback originator's control over when and if their failed callback should be retried.

BVSS

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

Recommendation

It is recommended to consider:

- Adding functionality to allow callback originators to cancel their pending callbacks when they're no longer needed or desired.
- Implementing a time-based expiration mechanism to prevent stale callbacks from persisting indefinitely.
- Adding access controls to `retryCallback()` so only authorized parties (callback originators or their delegates) can retry execution.

Remediation Comment

SOLVED: The **Ten Protocol team** solved this finding in commit `fdc1a72` by following the mentioned recommendation.

Remediation Hash

<https://github.com/ten-protocol/go-ten/commit/fdc1a7261852e33e7818c28bdad391e34dbfa7e7>

7.3 INADEQUATE CALLBACK ARRAY MANAGEMENT

// LOW

Description

The `TransactionPostProcessor` contract has two related issues with its `onBlockEndListeners` array management:

1. No mechanism exists to remove callbacks once added, even if they become compromised or malfunction.
2. The array can grow without bounds since callbacks can only be added, not removed.
3. The `addOnBlockEndCallback()` function lacks validation to prevent duplicate callbacks, allowing the same callback to be registered multiple times.

These issues combine to create a vulnerability where:

- Compromised or buggy callbacks cannot be removed and will continue to execute.
- The array will only grow over time, steadily increasing gas costs for the `onBlock()` function. Eventually, this could lead to transactions hitting block gas limits, causing a denial of service.

BVSS

AO:S/AC:L/AX:L/R:N/S:U/C:N/A:C/I:M/D:N/Y:N (2.3)

Recommendation

Consider implementing a comprehensive callback management system that includes:

- Functionality to remove specific callbacks from the array when they're no longer needed or have security issues.
- A reasonable maximum limit on the total number of callbacks that can be registered.
- Validation to prevent duplicate callbacks from being added multiple times.

Remediation Comment

PARTIALLY SOLVED: The **Ten Protocol team** partially solved this finding in commit `COMMIT` by enabling the removal of callbacks and adding some input validation in `TransactionPostProcessor`. However, mechanisms to set a maximum limit on callbacks or prevent duplicate entries were not implemented.

Remediation Hash

<https://github.com/ten-protocol/go-ten/commit/fdc1a7261852e33e7818c28bdad391e34dbfa7e7>

7.4 SINGLE STEP OWNERSHIP TRANSFER PROCESS

// LOW

Description

Some of the contracts in scope inherit the `Ownable` or `OwnableUpgradeable` contract implementations from OpenZeppelin's library, which are used to restrict access to certain functions to the contract owner. The `Ownable` pattern allows the contract owner to transfer ownership to another address using the `transferOwnership()` function. However, the `transferOwnership()` function does not include a two-step process to transfer ownership.

Regarding this, it is crucial that the address to which ownership is transferred is verified to be active and willing to assume ownership responsibilities. Otherwise, the contract could be locked in a situation where it is no longer possible to make administrative changes to it.

Additionally, the `renounceOwnership()` function allows renouncing to the owner permission. Renouncing ownership before transferring it would result in the contract having no owner, eliminating the ability to call privileged functions.

BVSS

AO:S/AC:L/AX:L/R:N/S:U/C:N/A:C/I:N/D:N/Y:N (2.0)

Recommendation

Consider using OpenZeppelin's `Ownable2Step` or `Ownable2StepUpgradeable` contracts over the `Ownable` and the `OwnableUpgradeable` implementations. `Ownable2Step` provides a two-step ownership transfer process, which adds an extra layer of security to prevent accidental ownership transfers.

Additionally, it is recommended that the owner cannot call the `renounceOwnership()` function to avoid losing ownership of the contract.

Remediation Comment

PARTIALLY SOLVED: The **Ten Protocol team** partially solved this finding in commit `fdc1a72` by implementing a two-step ownership transfer process in the `Fees` contract, but other ownable contracts like `TENToken` and `ZenBase` still use a single-step process.

Remediation Hash

<https://github.com/ten-protocol/go-ten/commit/fdc1a7261852e33e7818c28bdad391e34dbfa7e7>

7.5 FIXED TOKEN MINTING REGARDLESS OF TRANSACTION VALUE

// LOW

Description

The **ZenBase** contract's **onBlockEnd()** function mints exactly 1 token for each processed transaction, regardless of the transaction's value, complexity, or significance.

```
38 function onBlockEnd(Structs.Transaction[] calldata transactions) external onlyCaller {
39     if (transactions.length == 0) {
40         revert("No transactions to convert");
41     }
42     // Implement custom logic here
43     for (uint256 i=0; i<transactions.length; i++) {
44         // Process transactions
45         _mint(transactions[i].from, 1);
46         // emit TransactionProcessed(transactions[i].from, 1);
47     }
48 }
```

Since this approach fails to consider the relative importance of transactions, it could be exploited by users who deliberately create numerous small transactions to maximize their token rewards, which could lead to an economically unbalanced system where minor transactions receive the same reward as more substantial ones.

BVSS

AO:S/AC:L/AX:L/R:N/S:U/C:N/A:N/I:C/D:N/Y:N (2.0)

Recommendation

Consider implementing a more economically balanced token distribution system that considers transaction attributes.

Remediation Comment

RISK ACCEPTED: The **Ten Protocol team** made a business decision to accept the risk of this finding and not alter the contracts.

7.6 LACK OF TRANSACTION PROCESSING TRACKING MECHANISM

// LOW

Description

The `onBlockEnd()` function of the `ZenBase` contract lacks a mechanism to track which transactions have been processed, potentially allowing the same transaction to be processed multiple times. This could lead to unauthorized token minting through transaction replay attacks, undermining the token's economic model and distribution integrity.

```
38 function onBlockEnd(Structs.Transaction[] calldata transactions) external onlyCaller {
39     if (transactions.length == 0) {
40         revert("No transactions to convert");
41     }
42     // Implement custom logic here
43     for (uint256 i=0; i<transactions.length; i++) {
44         // Process transactions
45         _mint(transactions[i].from, 1);
46         // emit TransactionProcessed(transactions[i].from, 1);
47     }
48 }
```

Without a tracking mechanism, there's no way to prevent the same transaction from being submitted multiple times, potentially resulting in duplicate token minting.

BVSS

AO:S/AC:L/AX:L/R:N/S:U/C:N/A:N/I:C/D:N/Y:N (2.0)

Recommendation

Consider implementing a transaction tracking mechanism that records completed transactions in order to prevent duplicate processing and potential unauthorized token minting.

Remediation Comment

RISK ACCEPTED: The **Ten Protocol team** made a business decision to accept the risk of this finding and not alter the contracts.

7.7 MISSING ERROR HANDLING FOR CALLBACKS

// LOW

Description

The `onBlock()` function lacks error handling when executing callbacks. If any callback in the loop reverts, the entire transaction processing fails, creating a critical single point of failure.

```
46 function onBlock(Structs.Transaction[] calldata transactions) public onlySelf {
47     if (transactions.length == 0) {
48         revert("No transactions to convert");
49     }
50
51     // emit TransactionsConverted(transactions.length);
52
53     for (uint256 i = 0; i < onBlockEndListeners.length; ++i) {
54         IOnBlockEndCallback callback = onBlockEndListeners[i];
55         callback.onBlockEnd(transactions);
56     }
57 }
```

This design allows malicious or buggy callbacks to cause denial of service for the entire transaction post-processing system.

BVSS

AO:S/AC:L/AX:L/R:N/S:U/C:N/A:C/I:N/D:N/Y:N (2.0)

Recommendation

Consider implementing try-catch blocks around each callback invocation to isolate failures and handle them accordingly.

Remediation Comment

RISK ACCEPTED: The **Ten Protocol team** made a business decision to accept the risk of this finding and not alter the contracts.

7.8 UNNECESSARY EXTERNAL CALL IN _INTERNALREFUND WHEN REFUND AMOUNT IS ZERO

// INFORMATIONAL

Description

In the `PublicCallbacks` contract, the `_executeNextCallback()` function unconditionally calls the `_internalRefund` function regardless of whether there is actually any gas to refund. When `gasRefundValue` is 0, this results in an unnecessary external call that consumes gas without providing any benefit.

BVSS

AO:A/AC:L/AX:M/R:N/S:U/C:N/A:N/I:L/D:N/Y:N (1.7)

Recommendation

Add a check to avoid the external call when the refund amount is zero.

Remediation Comment

ACKNOWLEDGED: The **Ten Protocol team** made a business decision to acknowledge this finding and not alter the contracts.

7.9 MISSING INPUT VALIDATION

// INFORMATIONAL

Description

Throughout the codebase, there are several instances where input values are assigned without proper validation. For example, ensuring that an input address is not the zero address or that an integer falls within a valid range.

Failing to validate inputs before assigning them to state variables can lead to unexpected system behavior or even complete failure.

Instances of this issue include:

- In `Fees.initialize()`, `flatFee` is not validated to ensure it's within reasonable bounds before being assigned to `_messageFee`.
- In `Fees.setMessageFee()`, `newFeeForMessage` is not validated to ensure it's within reasonable bounds.
- In `SystemDeployer.constructor()`, the `remoteBridgeAddress` is not checked against the zero address before assignment.
- In `TransactionPostProcessor.addOnBlockEndCallback()`, callback addresses are added without verifying that:
 - They are not zero addresses.
 - They are actually contract addresses.
 - They properly implement the `IOOnBlockEndCallback` interface.

BVSS

AO:S/AC:L/AX:L/R:N/S:U/C:N/A:N/I:H/D:N/Y:N (1.5)

Recommendation

Add proper validation to ensure that the input values are within expected ranges and that addresses are not the zero address. This will help prevent unexpected behavior and improve the overall robustness of the code.

Remediation Comment

PARTIALLY SOLVED: The **Ten Protocol team** partially solved this finding in commit `fdc1a72` by adding several input validations, such as zero-address checks in `SystemDeployer.sol`, `ZenBase.sol`, and `TransactionPostProcessor.sol`. However, validations for fee-related parameters in the `Fees` contract were not implemented.

Remediation Hash

7.10 USE OF LOW-LEVEL TRANSFER METHOD

// INFORMATIONAL

Description

In the **Fees** contract, there is an instance of the **transfer()** method to withdraw native assets (e.g. Ether) from the smart contracts. Although the use of transfer has been a standard practice for sending native assets due to its built-in reentrancy protection (since it only forwards 2300 gas, preventing called contracts from performing state changes), it is not considered best practice.

The gas cost of EVM instructions may change significantly during hard forks, which may break already deployed contract systems that make fixed assumptions about gas costs.

BVSS

AO:S/AC:L/AX:L/R:N/S:U/C:N/A:M/I:N/D:N/Y:N (1.0)

Recommendation

Transfer native assets via the low-level **call()** method instead.

Remediation Comment

SOLVED: The **Ten Protocol team** solved this finding in commit **fdc1a72** by following the mentioned recommendation.

Remediation Hash

<https://github.com/ten-protocol/go-ten/commit/fdc1a7261852e33e7818c28bdad391e34dbfa7e7>

7.11 FLOATING PRAGMA

// INFORMATIONAL

Description

The contracts in scope currently use different floating pragma versions `^0.8.0`, `^0.8.22` and `^0.8.28` which means that the code can be compiled by any compiler version that is greater than these versions, and less than `0.9.0`.

However, it is recommended that contracts should be deployed with the same compiler version and flags used during development and testing. Locking the pragma helps to ensure that contracts do not accidentally get deployed using another pragma. For example, an outdated pragma version might introduce bugs that affect the contract system negatively.

Additionally, from Solidity versions `0.8.20` through `0.8.24`, the default target EVM version is set to `Shanghai`, which results in the generation of bytecode that includes `PUSH0` opcodes. Starting with version `0.8.25`, the default EVM version shifts to `Cancun`, introducing new opcodes for transient storage, `TSTORE` and `TLOAD`.

In this aspect, it is crucial to select the appropriate EVM version when it's intended to deploy the contracts on networks other than the Ethereum mainnet, which may not support these opcodes. Failure to do so could lead to unsuccessful contract deployments or transaction execution issues.

BVSS

AO:A/AC:L/AX:H/R:N/S:U/C:N/A:L/I:L/D:N/Y:N (1.0)

Recommendation

Lock the pragma version to the same version used during development and testing (for example: `pragma solidity 0.8.28;`), and make sure to specify the target EVM version when using Solidity versions from `0.8.20` and above if deploying to chains that may not support newly introduced opcodes.

Additionally, it is crucial to stay informed about the opcode support of different chains to ensure smooth deployment and compatibility.

Remediation Comment

ACKNOWLEDGED: The **Ten Protocol team** made a business decision to acknowledge this finding and not alter the contracts. They updated the Solidity compiler version specified in the contract pragmas, but did not uniformly lock all pragmas to a single, exact compiler version (without a caret) across all files.

7.12 USE OF REVERT STRINGS INSTEAD OF CUSTOM ERRORS

// INFORMATIONAL

Description

Throughout the files in scope, there are several instances where revert strings are used over custom errors.

In Solidity, replacing hard-coded revert message strings with the `Error()` syntax is an optimization strategy that can significantly reduce gas costs. Hard-coded strings, stored on the blockchain, increase the size and cost of deploying and executing contracts.

The `Error()` syntax allows for the definition of reusable, parameterized custom errors, leading to a more efficient use of storage and reduced gas consumption. This approach not only optimizes gas usage during deployment and interaction with the contract but also enhances code maintainability and readability by providing clearer, context-specific error information.

BVSS

AO:A/AC:H/AX:L/R:N/S:U/C:N/A:N/I:N/D:L/Y:N (0.8)

Recommendation

Consider replacing all revert strings with custom errors. For example:

```
error ConditionNotMet();  
if (!condition) revert ConditionNotMet();
```

or starting from Solidity `0.8.27` :

```
require(condition, ConditionNotMet());
```

For more references, [see here](#) and [here](#).

Remediation Comment

ACKNOWLEDGED: The **Ten Protocol team** made a business decision to acknowledge this finding and not alter the contracts.

7.13 UNHANDLED RETURN VALUE IN _PAYFORCALLBACK FUNCTION

// INFORMATIONAL

Description

In the `PublicCallbacks` contract, the `_payForCallback` function makes a low-level call to transfer ETH to the block's coinbase address without checking the return value of this operation. Although the function contains comments indicating that the success of this operation is not a concern, ignoring the return value could lead to silent failures in payment processing.

BVSS

AO:A/AC:L/AX:H/R:N/S:U/C:N/A:N/I:L/D:N/Y:N (0.8)

Recommendation

Consider modifying the function to at least monitor for failures, even if the contract chooses not to revert.

Remediation Comment

ACKNOWLEDGED: The **Ten Protocol team** made a business decision to acknowledge this finding and not alter the contracts.

7.14 MISSING EVENTS

// INFORMATIONAL

Description

Throughout the contracts in scope, there are several instances where administrative functions change contract state by modifying core state variables without them being reflected in event emissions.

The absence of events may hamper effective state tracking in off-chain monitoring systems.

Instances of this issue can be found in:

- `Fees.setMessageFee()`
- `Fees.withdrawalCollectedFees()`
- `PublicCallbacks.register()`
- `PublicCallbacks.executeNextCallback()`
- `TransactionPostProcessor.addOnBlockEndCallback()`

BVSS

AO:A/AC:L/AX:L/R:F/S:U/C:N/A:N/I:L/D:N/Y:N (0.6)

Recommendation

Emit events for all state changes that occur as a result of administrative functions to facilitate off-chain monitoring of the system.

Remediation Comment

SOLVED: The **Ten Protocol team** solved this finding in commit `fdc1a72` by following the mentioned recommendation.

Remediation Hash

<https://github.com/ten-protocol/go-ten/commit/fdc1a7261852e33e7818c28bdad391e34dbfa7e7>

7.15 OPEN TO-DOS

// INFORMATIONAL

Description

Throughout the contracts in scope, there are instances of open TODO comments that indicate incomplete documentation or potentially unfinished implementation.

These unresolved items create uncertainty about the completeness of the codebase and may hide critical implementation gaps that could lead to unexpected behavior or security vulnerabilities when the protocol is deployed.

BVSS

AO:A/AC:H/AX:L/R:F/S:U/C:N/A:N/I:M/D:N/Y:N (0.4)

Recommendation

Ensure that the TO-DO comments are implemented and resolved before the final release of the project.

Remediation Comment

PARTIALLY SOLVED: The **Ten Protocol team** partially solved this finding in commit **fdc1a72** by addressing most **TODO** comments and providing updated documentation or removing them. However, one **TODO** comment regarding an explanation remains in **Fees.sol**.

Remediation Hash

<https://github.com/ten-protocol/go-ten/commit/fdc1a7261852e33e7818c28bdad391e34dbfa7e7>

7.16 REDUNDANT DEFAULT VALUE ASSIGNMENT

// INFORMATIONAL

Description

In the `PublicCallbacks` contract's `initialize()` function, the state variables `nextCallbackId` and `lastUnusedCallbackId` are explicitly initialized to zero.

However, in Solidity, all state variables are automatically initialized to their default values when a contract is deployed - zero for integers, false for booleans, empty string for strings, etc. Therefore, explicitly initializing these `uint256` variables to zero is redundant and wastes gas during contract initialization.

BVSS

AO:A/AC:H/AX:H/R:N/S:U/C:N/A:N/I:N/D:L/Y:N (0.3)

Recommendation

Remove the redundant assignments from the `initialize()` function. If the function becomes empty after removing redundant assignments, consider whether the `initialize()` function is needed at all, or if it will be needed in the future for additional initialization logic when upgrading the contract.

Remediation Comment

SOLVED: The **Ten Protocol team** solved this finding in commit `fdc1a72` by following the mentioned recommendation.

Remediation Hash

<https://github.com/ten-protocol/go-ten/commit/fdc1a7261852e33e7818c28bdad391e34dbfa7e7>

7.17 PUBLIC FUNCTIONS CAN BE MARKED EXTERNAL

// INFORMATIONAL

Description

Some functions throughout the contracts in scope are currently defined with the `public` visibility modifier, even though the functions are not called from within the contract, resulting in higher gas costs than necessary.

Instances of this issue include:

- `TransactionPostProcessor.addOnBlockEndCallback()`
- `TransactionPostProcessor.onBlock()`
- `TENToken.pause()`
- `TENToken.unpause()`

BVSS

AO:A/AC:H/AX:H/R:N/S:U/C:N/A:N/I:L/D:N/Y:N (0.3)

Recommendation

Modify the `public` functions not used within the contracts with the `external` visibility modifier.

Remediation Comment

SOLVED: The **Ten Protocol team** solved this finding in commit `fdc1a72` by following the mentioned recommendation.

Remediation Hash

<https://github.com/ten-protocol/go-ten/commit/fdc1a7261852e33e7818c28bdad391e34dbfa7e7>

7.18 COMMENTED FUNCTIONALITY

// INFORMATIONAL

Description

Throughout the contracts in scope, there are instances of commented out code that is not used. This code may introduce unnecessary confusion to the contract.


While commenting out code can be useful for debugging or testing purposes, it can also lead to confusion and make the codebase harder to maintain.

BVSS

AO:A/AC:L/AX:H/R:F/S:U/C:N/A:N/I:L/D:N/Y:N (0.2)

Recommendation

Remove the commented-out lines of code to clean up the contract and improve readability.

Alternatively, if the functionality is  needed, uncomment and update the code as necessary.

Remediation Comment

PARTIALLY SOLVED: The **Ten Protocol team** partially solved this finding in commit **fdc1a72** by removing the commented-out event emission in **TransactionPostProcessor.sol**, but a commented-out event emission remains in **ZenBase.sol**.

Remediation Hash

<https://github.com/ten-protocol/go-ten/commit/fdc1a7261852e33e7818c28bdad391e34dbfa7e7>

7.19 MISSING VARIABLE VISIBILITY

// INFORMATIONAL

Description

The `onBlockEndListeners` variable of the `TransactionPostProcessor` contract is not explicitly defined. By default, variables are set with the `internal` visibility.

However, it is considered best practice to explicitly specify visibility to enhance clarity and prevent ambiguity. Clearly labeling the visibility of all variables and functions will help in maintaining clear and understandable code.

BVSS

AO:A/AC:H/AX:L/R:F/S:U/C:N/A:L/I:N/D:N/Y:N (0.2)

Recommendation

Explicitly define the visibility of all variables in the contracts to enhance readability and reduce the potential for errors.

Remediation Comment

SOLVED: The **Ten Protocol team** solved this finding in commit `fdc1a72` by following the mentioned recommendation.

Remediation Hash

<https://github.com/ten-protocol/go-ten/commit/fdc1a7261852e33e7818c28bdad391e34dbfa7e7>

7.20 LACK OF NAMED MAPPINGS

// INFORMATIONAL

Description

The project contains several unnamed mappings despite using a Solidity version that supports named mappings.

Named mappings improve code readability and self-documentation by explicitly stating their purpose.

Instances of this issue can be found in:

- `PublicCallbacks.callbacks()`
- `PublicCallbacks.callbackBlockNumber()`

BVSS

AO:A/AC:H/AX:H/R:F/S:U/C:N/A:N/I:L/D:N/Y:N (0.1)

Recommendation

Consider refactoring the mappings to use named arguments, which will enhance code readability and make the purpose of each mapping more explicit. For example:

```
mapping(address myAddress => bool myBool) public myMapping;
```

Remediation Comment

SOLVED: The **Ten Protocol team** solved this finding in commit `fdc1a72` by following the mentioned recommendation.

Remediation Hash

<https://github.com/ten-protocol/go-ten/commit/fdc1a7261852e33e7818c28bdad391e34dbfa7e7>

7.21 UNUSED IMPORT

// INFORMATIONAL

Description

In the **TransactionPostProcessor** contract, there is an import that is not used within the contract and that can be removed to improve code readability and maintainability:

```
import "@openzeppelin/contracts/token/ERC20/utils/SafeERC20.sol";
```

BVSS

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

Recommendation

Remove the unused import from the contract.

Remediation Comment

SOLVED: The **Ten Protocol team** solved this finding in commit **fdc1a72** by following the mentioned recommendation.

Remediation Hash

<https://github.com/ten-protocol/go-ten/commit/fdc1a7261852e33e7818c28bdad391e34dbfa7e7>

7.22 UNOPTIMIZED FOR LOOPS

// INFORMATIONAL

Description

Throughout the code in scope, there are several instances of unoptimized for loop declarations that incur higher gas costs than necessary. These inefficiencies typically manifest as array length calculations in each iteration, unnecessary storage variable reads or updates, lack of unchecked increments for counters that cannot overflow, amongst others.

Such patterns significantly increase gas consumption during execution, particularly for functions that process large arrays or are called frequently. This results in higher transaction costs for users interacting with the protocol and, in extreme cases, could lead to functions reaching the block gas limit, effectively causing denial of service when the protocol handles larger datasets.

BVSS

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

Recommendation

Optimize the `for` loop declarations to reduce gas costs. Best practices include:

- The non-redundant initialization of the iterator with a default value (declaring simply `i` is equivalent to `i = 0` but more gas efficient).
- The use of the pre-increment operator (inside an `unchecked` block if using Solidity `>=0.8.0` and `<=0.8.21`: `unchecked { ++i }`, or simply `++i` if compiling with Solidity `>=0.8.22`).

Additionally, when reading from storage variables, it is recommended to reduce gas costs by caching the array to read locally and iterate over it to avoid reading from storage on every iteration.

For example:

- In Solidity versions between `>=0.8.0` and `<=0.8.21`:

```
uint256[] memory arrayInMemory = arrayInStorage;
for (uint256; i < arrayInMemory.length ; ) {
    // code logic
    unchecked { ++i; }
}
```

- In Solidity versions `>=0.8.22`:

```
uint256[] memory arrayInMemory = arrayInStorage;
for (uint256; i < arrayInMemory.length ; ++i) {
```

```
// code logic  
}
```

Remediation Comment

SOLVED: The **Ten Protocol team** solved this finding in commit **fdc1a72** by following the mentioned recommendation.

Remediation Hash

<https://github.com/ten-protocol/go-ten/commit/fdc1a7261852e33e7818c28bdad391e34dbfa7e7>

8. AUTOMATED TESTING

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.

The security team assessed all findings identified by the Slither software, however, findings with related to external dependencies are not included in the below results for the sake of report readability.

Output

The findings obtained as a result of the Slither scan were reviewed, and some were not included in the report because they were determined as false positives.

```
Reentrancy in PublicCallbacks.executeNextCallback() (src/system/contracts/PublicCallbacks.sol#110-137):
  External calls:
  - (success,None) = callback.target.call(gas: prepaidGas)(callback.data) (src/system/contracts/PublicCallbacks.sol#119)
  State variables written after the call(s):
  - popCurrentCallback() (src/system/contracts/PublicCallbacks.sol#131)
  - delete callback[lastUnusedCallbackId] (src/system/contracts/PublicCallbacks.sol#98)
  PublicCallbacks.callbacks (src/system/contracts/PublicCallbacks.sol#32) can be used in cross function reentrancies:
  - PublicCallbacks.addCallback(address,bytes,uint256) (src/system/contracts/PublicCallbacks.sol#87-91)
  - PublicCallbacks.callbacks (src/system/contracts/PublicCallbacks.sol#32)
  - PublicCallbacks.getCurrentCallbackToExecute() (src/system/contracts/PublicCallbacks.sol#93-95)
  - PublicCallbacks.popCurrentCallback() (src/system/contracts/PublicCallbacks.sol#97-100)
  - PublicCallbacks.reattemptCallback(uint256) (src/system/contracts/PublicCallbacks.sol#65-72)
  - moveToNextCallback() (src/system/contracts/PublicCallbacks.sol#133)
  - lastUnusedCallbackId ++ (src/system/contracts/PublicCallbacks.sol#163)
  PublicCallbacks.lastUnusedCallbackId (src/system/contracts/PublicCallbacks.sol#34) can be used in cross function reentrancies:
  - PublicCallbacks.executeNextCallback() (src/system/contracts/PublicCallbacks.sol#110-137)
  - PublicCallbacks.executeNextCallbacks() (src/system/contracts/PublicCallbacks.sol#77-81)
  - PublicCallbacks.getCurrentCallbackToExecute() (src/system/contracts/PublicCallbacks.sol#93-95)
  - PublicCallbacks.initialize() (src/system/contracts/PublicCallbacks.sol#45-49)
  - PublicCallbacks.moveToNextCallback() (src/system/contracts/PublicCallbacks.sol#182-184)
  - PublicCallbacks.popCurrentCallback() (src/system/contracts/PublicCallbacks.sol#97-100)
  - PublicCallbacks.reattemptCallback(uint256) (src/system/contracts/PublicCallbacks.sol#65-72):
Reentrancy in PublicCallbacks.reattemptCallback(uint256) (src/system/contracts/PublicCallbacks.sol#65-72):
  External calls:
  - (success,None) = callback.target.call(callback.data) (src/system/contracts/PublicCallbacks.sol#67)
  State variables written after the call(s):
  - delete callbackBlockNumber[callbackId] (src/system/contracts/PublicCallbacks.sol#70)
  PublicCallbacks.callbackBlockNumber (src/system/contracts/PublicCallbacks.sol#36) can be used in cross function reentrancies:
  - PublicCallbacks.addCallback(address,bytes,uint256) (src/system/contracts/PublicCallbacks.sol#87-91)
  - PublicCallbacks.callbackBlockNumber (src/system/contracts/PublicCallbacks.sol#36)
  - PublicCallbacks.callbackBlockNumber (src/system/contracts/PublicCallbacks.sol#40-43)
  - PublicCallbacks.callbackBlockNumber (src/system/contracts/PublicCallbacks.sol#40-43)
  - PublicCallbacks.popCurrentCallback() (src/system/contracts/PublicCallbacks.sol#97-100)
  - PublicCallbacks.reattemptCallback(uint256) (src/system/contracts/PublicCallbacks.sol#65-72)
  - delete callback[callbackId] (src/system/contracts/PublicCallbacks.sol#69)
  PublicCallbacks.callbacks (src/system/contracts/PublicCallbacks.sol#32) can be used in cross function reentrancies:
  - PublicCallbacks.addCallback(address,bytes,uint256) (src/system/contracts/PublicCallbacks.sol#87-91)
  - PublicCallbacks.callbacks (src/system/contracts/PublicCallbacks.sol#32)
  - PublicCallbacks.getCurrentCallbackToExecute() (src/system/contracts/PublicCallbacks.sol#93-95)
  - PublicCallbacks.popCurrentCallback() (src/system/contracts/PublicCallbacks.sol#97-100)
  - PublicCallbacks.reattemptCallback(uint256) (src/system/contracts/PublicCallbacks.sol#65-72)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#reentrancy-vulnerabilities-1
INFO:Detectors:
PublicCallbacks.payForCallback(uint256) (src/system/contracts/PublicCallbacks.sol#151-159) ignores return value by block.coinbase.call{value: gasPayment}() (src/system/contracts/PublicCallbacks.sol#158)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#unchecked-low-level-calls
INFO:Detectors:
Fees.setMessageFee(uint256) (src/system/contracts/Fees.sol#36-39) should emit an event for:
  - _messageFee = newFeeForMessage (src/system/contracts/Fees.sol#37)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#missing-events-arithmetic
INFO:Detectors:
PublicCallbacks.executeNextCallback() (src/system/contracts/PublicCallbacks.sol#110-137) has external calls inside a loop: (success,None) = callback.target.call(gas: prepaidGas)(callback.data) (src/system/contracts/PublicCallbacks.sol#119)
PublicCallbacks.internalRefund(uint256,address,uint256) (src/system/contracts/PublicCallbacks.sol#139-149) has external calls inside a loop: (success,None) = to.call(gas: 45000,value: gasRefund)(data) (src/system/contracts/PublicCallbacks.sol#144)
PublicCallbacks.payForCallback(uint256) (src/system/contracts/PublicCallbacks.sol#151-159) has external calls inside a loop: block.coinbase.call{value: gasPayment}() (src/system/contracts/PublicCallbacks.sol#158)
TransactionPostProcessor.onBlock(Structs.Transaction[]) (src/system/contracts/TransactionPostProcessor.sol#51-68) has external calls inside a loop: callback.onBlockEnd(transactions) (src/system/contracts/TransactionPostProcessor.sol#66)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#oncall-inside-a-loop
INFO:Detectors:
Reentrancy in PublicCallbacks.executeNextCallback() (src/system/contracts/PublicCallbacks.sol#110-137):
  External calls:
  - (success,None) = callback.target.call(gas: prepaidGas)(callback.data) (src/system/contracts/PublicCallbacks.sol#119)
  State variables written after the call(s):
  - popCurrentCallback() (src/system/contracts/PublicCallbacks.sol#131)
  - delete callbackBlockNumber[lastUnusedCallbackId] (src/system/contracts/PublicCallbacks.sol#99)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#reentrancy-vulnerabilities-2
INFO:Detectors:
```



```

Reentrancy in SystemDeployer.deployAnalyze(address) (src/system/contracts/SystemDeployer.sol#38-43):
  External calls:
    - transactionsPostProcessorProxy = deployProxy(address(transactionsPostProcessor), eoaAdmin, callData) (src/system/contracts/SystemDeployer.sol#41)
    - proxy = new TransparentUpgradeableProxy(_logic, _admin, _data) (src/system/contracts/SystemDeployer.sol#34)
  Event emitted after the call(s):
    - SystemContractDeployed(transactionsPostProcessor, transactionsPostProcessorProxy) (src/system/contracts/SystemDeployer.sol#42)
Reentrancy in SystemDeployer.deployCrossChainMessenger(address, address) (src/system/contracts/SystemDeployer.sol#68-74):
  External calls:
    - crossChainMessengerProxy = deployProxy(address(crossChainMessenger), eoaAdmin, callData) (src/system/contracts/SystemDeployer.sol#71)
    - proxy = new TransparentUpgradeableProxy(_logic, _admin, _data) (src/system/contracts/SystemDeployer.sol#34)
  Event emitted after the call(s):
    - SystemContractDeployed(crossChainMessenger, crossChainMessengerProxy) (src/system/contracts/SystemDeployer.sol#72)
Reentrancy in SystemDeployer.deployEthereumBridge(address, address, address) (src/system/contracts/SystemDeployer.sol#76-81):
  External calls:
    - ethereumBridgeProxy = deployProxy(address(ethereumBridge), eoaAdmin, callData) (src/system/contracts/SystemDeployer.sol#79)
    - proxy = new TransparentUpgradeableProxy(_logic, _admin, _data) (src/system/contracts/SystemDeployer.sol#34)
  Event emitted after the call(s):
    - SystemContractDeployed(ethereumBridge, ethereumBridgeProxy) (src/system/contracts/SystemDeployer.sol#80)
Reentrancy in SystemDeployer.deployFees(address, uint256) (src/system/contracts/SystemDeployer.sol#45-51):
  External calls:
    - feesProxy = deployProxy(address(fees), eoaAdmin, callData) (src/system/contracts/SystemDeployer.sol#48)
    - proxy = new TransparentUpgradeableProxy(_logic, _admin, _data) (src/system/contracts/SystemDeployer.sol#34)
  Event emitted after the call(s):
    - SystemContractDeployed(fees, feesProxy) (src/system/contracts/SystemDeployer.sol#49)
Reentrancy in SystemDeployer.deployMessageBus(address, address) (src/system/contracts/SystemDeployer.sol#53-59):
  External calls:
    - messageBusProxy = deployProxy(address(messageBus), eoaAdmin, callData) (src/system/contracts/SystemDeployer.sol#56)
    - proxy = new TransparentUpgradeableProxy(_logic, _admin, _data) (src/system/contracts/SystemDeployer.sol#34)
  Event emitted after the call(s):
    - SystemContractDeployed(messageBus, messageBusProxy) (src/system/contracts/SystemDeployer.sol#57)
Reentrancy in SystemDeployer.deployPublicCallbacks(address) (src/system/contracts/SystemDeployer.sol#61-66):
  External calls:
    - publicCallbacksProxy = deployProxy(address(publicCallbacks), eoaAdmin, callData) (src/system/contracts/SystemDeployer.sol#64)
    - proxy = new TransparentUpgradeableProxy(_logic, _admin, _data) (src/system/contracts/SystemDeployer.sol#34)
  Event emitted after the call(s):
    - SystemContractDeployed(publicCallbacks, publicCallbacksProxy) (src/system/contracts/SystemDeployer.sol#65)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#reentrancy-vulnerabilities-3
INFO:Detectors:
PublicCallbacks.executeNextCallback() (src/system/contracts/PublicCallbacks.sol#110-137) tries to limit the gas of an external call that controls implicit decoding
  (success, None) = callback.target.call{gas: prepaidGas}(callback.data) (src/system/contracts/PublicCallbacks.sol#119)
PublicCallbacks.internalRefund(uint256, address, uint256) (src/system/contracts/PublicCallbacks.sol#139-149) tries to limit the gas of an external call that controls implicit decoding
  (success, None) = to.call{gas: 45000, value: gasRefund}(data) (src/system/contracts/PublicCallbacks.sol#144)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#return-bomb
INFO:Detectors:
PublicCallbacks.popCurrentCallback() (src/system/contracts/PublicCallbacks.sol#97-100) has costly operations inside a loop:
  - delete callbacks[lastUnusedCallbackId] (src/system/contracts/PublicCallbacks.sol#98)
PublicCallbacks.popCurrentCallback() (src/system/contracts/PublicCallbacks.sol#97-100) has costly operations inside a loop:
  - delete callbackBlockNumber[lastUnusedCallbackId] (src/system/contracts/PublicCallbacks.sol#99)
PublicCallbacks.moveToNextCallback() (src/system/contracts/PublicCallbacks.sol#102-104) has costly operations inside a loop:
  - lastUnusedCallbackId ++ (src/system/contracts/PublicCallbacks.sol#103)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#costly-operations-inside-a-loop
INFO:Detectors:
PublicCallbacks (src/system/contracts/PublicCallbacks.sol#12-160) should inherit from IPublicCallbacks (src/system/interfaces/IPublicCallbacks.sol#8-21)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#missing-inheritance
INFO:Detectors:
TENToken.initialize(address, address) (src/ten_erc20/TENToken.sol#20-27) uses literals with too many digits:
  - _mint(recipient, 10000000000 * 10 ** decimals()) (src/ten_erc20/TENToken.sol#26)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#too-many-digits
INFO:Detectors:
ZenBase._caller (src/system/utis/ZenBase.sol#21) should be immutable
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#state-variables-that-could-be-declared-immutable

```

Halborn strongly recommends conducting a follow-up assessment of the project either within six months or immediately following any material changes to the codebase, whichever comes first. This approach is crucial for maintaining the project's integrity and addressing potential vulnerabilities introduced by code modifications.