// Private Smart Contract Security Assessment 04.22.2024 - 04.29.2024

InceptionLRT Bridge

Tagus Labs

HALBORN

InceptionLRT Bridge - Tagus Labs

Prepared by: HALBORN

Last Updated 05/16/2024

Date of Engagement by: April 22nd, 2024 - April 29th, 2024

Summary

100% ⓘ OF ALL REPORTED FINDINGS HAVE BEEN ADDRESSED

ALL FINDINGS	CRITICAL	HIGH	MEDIUM	LOW	INFORMATIONAL
15	0	0	0	5	10

TABLE OF CONTENTS

1. Introduction

Tagus Labs engaged `Halborn` to conduct a security assessment on their smart contracts beginning on 04-22-2024 and ending on 04-29-2024. The security assessment was scoped to the smart contracts provided in the https://github.com/inceptionlrt/bridge/tree/feat/extend-bridge-with-xerc20 GitHub repository. Commit hashes and further details can be found in the Scope section of this report.

2. Assessment Summary

Halborn was provided 1 week 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 mostly addressed by the `Tagus Labs team`. The main identified issues are:

- Lack of checks for deposits and withdrawals recipients may allow for tokens lost in `XERC20Lockbox` contract.
- Low limit amount for a bridge will result in a rate per second equal to zero.
- Possible denial of service for bridges when high limits are set.

3. Test Approach And 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 contracts' solidity 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 walk-through.
- Manual assessment of use and safety for the critical Solidity variables and functions in scope to identify any arithmetic-related vulnerability classes.
- Manual testing with custom scripts (`Foundry`).
- Static Analysis of security for scoped contract, and imported functions.

Out-Of-Scope

- External libraries and financial-related attacks.
- New features/implementations after/within the **remediation commit IDs**.
- The following files, while they affect the in-scope code, were not specifically reviewed as part of the scope and are assumed to not contain any issues:
    - lib/EthereumVerifier.sol
    - lib/ProofParser.sol
    - lib/CallDataRLPReader.sol
    - lib/Utils.sol

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:

EXPLOITABILIY 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

EXPLOITABILIY 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) Low (I:L) Medium (I:M) High (I:H) Critical (I:C)	0 0.25 0.5 0.75 1
Availability (A)	None (A:N) Low (A:L) Medium (A:M) High (A:H) Critical (A:C)	0 0.25 0.5 0.75 1
Deposit (D)	None (D:N) Low (D:L) Medium (D:M) High (D:H) Critical (D:C)	0 0.25 0.5 0.75 1
Yield (Y)	None (Y:N) Low (Y:L) Medium (Y:M) High (Y:H) Critical (Y:C)	0 0.25 0.5 0.75 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) 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

5. SCOPE

FILES AND REPOSITORY

(a) Repository: bridge

(b) Assessed Commit ID: 8281a3c

(c) Items in scope:

- contracts/bridge/InceptionBridge.sol
- contracts/XERC20/XERC20.sol
- contracts/XERC20/XERC20Lockbox.sol
- contracts/factory/BridgeFactory.sol
- contracts/proxy/InitializableTransparentUpgradeableProxy.sol

Out-of-Scope: lib/CallDataRLPReader.sol, lib/EthereumVerifier.sol, lib/ProofParser.sol, lib/Utils.sol

REMEDIATION COMMIT ID:

- a98d3e2a98d3e2

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

6. ASSESSMENT SUMMARY & FINDINGS OVERVIEW

CRITICAL	HIGH	MEDIUM	LOW	INFORMATIONAL
0	0	0	5	10

SECURITY ANALYSIS	RISK LEVEL	REMEDIATION DATE
LACK OF CHECKS FOR DEPOSITS AND WITHDRAWALS RECIPIENTS MAY ALLOW FOR TOKENS LOST IN XERC20LOCKBOX CONTRACT	Low	SOLVED - 04/28/2024
LOW LIMIT AMOUNT FOR A BRIDGE WILL RESULT IN A RATE PER SECOND EQUAL TO ZERO	Low	SOLVED - 04/28/2024

SECURITY ANALYSIS	RISK LEVEL	REMEDIATION DATE
POSSIBLE DENIAL OF SERVICE FOR BRIDGES WHEN HIGH LIMITS ARE SET	Low	RISK ACCEPTED
CENTRALIZATION RISKS DUE TO PRIVILEGED ACCESS BY OWNER	Low	RISK ACCEPTED
POTENTIAL RISKS IN SINGLE-STEP TRANSFER OF OWNERSHIP	Low	RISK ACCEPTED
LACK OF CHECK FOR ZERO ADDRESS WHEN ADDING DESTINATION	Informational	SOLVED - 04/28/2024
LACK OF CHECK FOR DEPOSITS AND WITHDRAWALS WITH ZERO AMOUNT	Informational	ACKNOWLEDGED
PUBLIC FUNCTIONS NOT CALLED WITHIN THE CONTRACT CAN BE MADE EXTERNAL	Informational	SOLVED - 04/28/2024
USE OF POST-INCREMENT OPERATOR	Informational	SOLVED - 04/28/2024
UNUSED IMPORT	Informational	SOLVED - 04/28/2024
MISSING ERROR DESCRIPTION	Informational	SOLVED - 04/28/2024
UNLOCKED PRAGMA COMPILERS	Informational	ACKNOWLEDGED

SECURITY ANALYSIS	RISK LEVEL	REMEDIATION DATE
PUSHO IS NOT SUPPORTED BY ALL CHAINS	Informational	ACKNOWLEDGED
INCEPTIONBRIDGE CONTRACT ASSUMES UNIFORM DECIMAL PLACES ACROSS TOKENS	Informational	ACKNOWLEDGED
USE OF FULL FILE IMPORTS	Informational	ACKNOWLEDGED

7. FINDINGS & TECH DETAILS

7.1 LACK OF CHECKS FOR DEPOSITS AND WITHDRAWALS RECIPIENTS MAY ALLOW FOR TOKENS LOST IN XERC20LOCKBOX CONTRACT

// LOW

Description

In the `XERC20Lockbox` contract, the functions that allow users to make deposits and withdrawals to an arbitrary recipient do not check if the recipient is the `XERC20Lockbox` contract itself. This may end up in self-transfers that result in tokens being locked in the contract.
Consider the following scenarios:

Scenario A
The function `depositNativeTo()` allows users to make native deposits and transfer the corresponding tokens to an arbitrary receiver. If the `_to` address is set (intentionally or accidentally) to the `XERC20Lockbox` address, it allows for `XERC20` tokens to be deposited to the `XERC20Lockbox` contract.

```
  function depositNativeTo(address _to) public payable {
    if (!IS_NATIVE) revert IXERC20Lockbox_NotNative();
    _deposit(_to, msg.value);
  }
```

```
  function _deposit(address _to, uint256 _amount) internal {
    if (!IS_NATIVE) ERC20.safeTransferFrom(msg.sender, address(this),
_amount);
    XERC20.mint(_to, _amount);
    emit Deposit(_to, _amount);
  }
```

Scenario B
Similarly, the function `depositTo` allows for users to make token deposits to an arbitrary recipient. In this case, the result will be the same as the previous scenario, with additional `ERC20` tokens being deposited into the contract.

```
  function depositTo(address _to, uint256 _amount) external {
    if (IS_NATIVE) revert IXERC20Lockbox_Native();
    _deposit(_to, _amount);
  }
```

Scenario C
Similarly, the `withdrawTo` function may result in the same outcome with additional steps, leaving `XERC20` tokens deposited to the contract.

```
function _withdraw(address _to, uint256 _amount) internal {
  XERC20.burn(msg.sender, _amount);
  if (IS_NATIVE) {
    (bool _success, ) = payable(_to).call{value: _amount}("");
    if (!_success) revert IXERC20Lockbox_WithdrawFailed();
  } else {
    ERC20.safeTransfer(_to, _amount);
  }
  emit Withdraw(_to, _amount);
}
```

If the `_to` address is set to the `XERC20Lockbox` address and the `IS_NATIVE` condition is met, execution will reach the code to transfer native assets:

```
(bool _success, ) = payable(_to).call{value: _amount}("");
if (!_success) revert IXERC20Lockbox_WithdrawFailed();
```

This code will effectively make a native transfer to the lockbox contract, which in turn will activate the execution of the local `receive()` function, executing `depositNative()`. This will make deposits on behalf of the `XERC20Lockbox` (being its address the `msg.sender`) and contract and `XERC20` tokens will be minted to it.

```
receive() external payable {
  depositNative();
}
```

Given these scenarios, the `XERC20` tokens will remain locked in the contract as there is no functionality in `XERC20Lockbox` to retrieve them.

Proof of Concept

Scenario A:

```
function test_depositNativeToLockbox(address receiver, uint256 amount) public {
  vm.assume(receiver != address(0));
  vm.assume(amount != 0);
  test_deployLockboxNative();

  receiver = address(lockbox);
  vm.deal(depositor, amount);
  vm.startPrank(depositor);

  uint256 balanceBefore = xerc20.balanceOf(receiver);
  lockbox.depositNativeTo{value: amount}(receiver);
  uint256 balanceAfter = xerc20.balanceOf(receiver);
  assertEq(balanceBefore + amount, balanceAfter);
```

```
    vm.stopPrank();
  }
```

```
Ran 1 test for test/BridgeFactoryTokenAndLockbox.t.sol:BridgeFactoryTokenAndLockboxTest
[PASS] test_depositNativeToLockbox(address,uint256) (runs: 26, μ: 2496923, ~: 2496923)
Suite result: ok. 1 passed; 0 failed; 0 skipped; finished in 33.82ms (31.28ms CPU time)

Ran 1 test suite in 59.67ms (33.82ms CPU time): 1 tests passed, 0 failed, 0 skipped (1 total tests)
```

Scenario B:

```
  function test_depositTokenToLockbox(address receiver, uint256 amount)
public {
    vm.assume(receiver != address(0));
    vm.assume(amount != 0);
    test_deployLockboxNotNative();

    receiver = address(lockbox);
    deal(address(mockToken), depositor, amount);
    vm.startPrank(depositor);
    mockToken.approve(address(lockbox), amount);

    uint256 balanceBefore = mockToken.balanceOf(receiver);
    lockbox.depositTo(receiver, amount);
    uint256 balanceAfter = mockToken.balanceOf(receiver);
    assertEq(balanceBefore + amount, balanceAfter);
    vm.stopPrank();
  }
```

```
Ran 1 test for test/BridgeFactoryTokenAndLockbox.t.sol:BridgeFactoryTokenAndLockboxTest
[PASS] test_depositTokenToLockbox(address,uint256) (runs: 26, μ: 2648759, ~: 2648080)
Suite result: ok. 1 passed; 0 failed; 0 skipped; finished in 28.89ms (26.87ms CPU time)

Ran 1 test suite in 41.14ms (28.89ms CPU time): 1 tests passed, 0 failed, 0 skipped (1 total tests)
```

```
  function test_withdrawToLockboxNative(uint256 withdrawAmount) public {
    test_deployLockboxNative();
    withdrawAmount = bound(withdrawAmount, 1, type(uint).max);
    address receiver = address(lockbox); //set lockbox as receiver

    vm.deal(receiver, withdrawAmount);

    vm.prank(address(lockbox));
    xerc20.mint(depositor, withdrawAmount);

    uint256 balanceXerc20Before = xerc20.balanceOf(receiver);
    vm.startPrank(depositor);

    lockbox.withdrawTo(receiver, withdrawAmount);
```

```
    uint256 balanceXerc20After = xerc20.balanceOf(receiver);
    assertEq(balanceXerc20Before + withdrawAmount, balanceXerc20After);
  }
```

```
Ran 1 test for test/BridgeFactoryTokenAndLockbox.t.sol:BridgeFactoryTokenAndLockboxTest
[PASS] test_withdrawToLockboxNative(uint256) (runs: 27, μ: 2510458, ~: 2510448)
Suite result: ok. 1 passed; 0 failed; 0 skipped; finished in 28.67ms (26.29ms CPU time)

Ran 1 test suite in 46.49ms (28.67ms CPU time): 1 tests passed, 0 failed, 0 skipped (1 total tests)
```

BVSS

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

Recommendation

Consider adding checks on the aforementioned functions to ensure that in no case the recipient of the transfers is the `XERC20Lockbox` contract itself.

Remediation Plan

SOLVED: The **Tagus Labs team** has addressed the finding in commit `a98d3e2` by following the mentioned recommendation of adding a check to verify that the receiver is not the `XERC20Lockbox` contract itself.

Remediation Hash

https://github.com/inceptionlrt/bridge/commit/a98d3e2ab68950083c82f453c6e01e4b1205a2d0

References

XERC20Lockbox.sol#L52-L55
XERC20Lockbox.sol#L62-L65
XERC20Lockbox.sol#L71-L74
XERC20Lockbox.sol#L89-L91
XERC20Lockbox.sol#L98-L107
XERC20Lockbox.sol#L114-L120
XERC20Lockbox.sol#L125-L127

7.2 LOW LIMIT AMOUNT FOR A BRIDGE WILL RESULT IN A RATE PER SECOND EQUAL TO ZERO

// LOW

Description

In the `changeBurnerLimit` and `changeMintingLimit` functions, which are executed via `setBridgeLimits()`, any limit set below `86400` (equal to `1 days` in Solidity) will result in a zero rate per second, affecting the possibility of dynamic limit within the `DURATION` threshold. This may occur because the rate per second is calculated as following in the aforementioned functions:

```
bridges[_bridge].minterParams.ratePerSecond = _limit / _DURATION;
```

```
bridges[_bridge].burnerParams.ratePerSecond = _limit / _DURATION;
```

where `uint256 private constant _DURATION = 1 days`.
This may result in a static limit for the specified bridge, as with a rate per second with `0` value will not change the `_calculatedLimit` value in the following code execution whenever the `_getCurrentLimit()` function is called:

```
uint256 _timePassed = block.timestamp - _timestamp;
uint256 _calculatedLimit = _limit + (_timePassed * _ratePerSecond);
_limit = _calculatedLimit > _maxLimit ? _maxLimit : _calculatedLimit;
```

BVSS

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

Recommendation

It is recommended to set a minimum threshold when invoking `setBridgeLimits()` or, alternatively, incorporate a validation step to confirm that `ratePerSecond` is not set to zero.

Remediation Plan

SOLVED: The **Tagus Labs team** has addressed the finding in commit `a98d3e2` by following the mentioned recommendation of adding a check for a minimum threshold.

Remediation Hash

https://github.com/inceptionlrt/bridge/commit/a98d3e2ab68950083c82f453c6e01e4b1205a2d0

References

XERC20.sol#L82-L90

7.3 POSSIBLE DENIAL OF SERVICE FOR BRIDGES WHEN HIGH LIMITS ARE SET

// LOW

Description

When using Solidity versions `0.8.0` and above, the compiler natively handles overflows and underflows by reverting the transaction if these issues occur. In the `XERC20` contract, specifically in the `_getCurrentLimit()` function, there's a potential for an unhandled overflow, which could cause the transaction to revert when calculating the current limit:

```
uint256 _calculatedLimit = _limit + (_timePassed * _ratePerSecond);
```

If the multiplication operation between `_timePassed` and `_ratePerSecond` results in a value greater than `type(uint256).max`, the transaction will revert. Similarly, if the result of the addition `_limit` and `(_timePassed * _ratePerSecond)` is greater than `type(uint256).max`, the transaction will also revert, effectively denying service for the bridge attempting to mint or burn `XERC20` tokens under these circumstances.

Proof of Concept

For example, even though the limit set is very high, a bridge is not able to mint 2 tokens due to the overflow occuring when calculating the new limit.

```
  function test_mintVeryHighLimitDOS(address _bridge, uint256 _burningLimit, uint256
amount) public {
    vm.assume(_burningLimit != 0);
    vm.assume(_bridge != address(0));
    amount = bound(amount, 1, type(uint256).max);
    test_setBridgeLimits(_bridge, type(uint256).max, _burningLimit);
    vm.startPrank(_bridge);

    uint256 currentLimitBefore = xerc20.mintingCurrentLimitOf(_bridge);
    uint256 maxLimitBefore = xerc20.mintingMaxLimitOf(_bridge);

    xerc20.mint(_bridge, 1);
    assertEq(xerc20.balanceOf(_bridge), 1);
    skip(1);
    vm.expectRevert();
    xerc20.mint(_bridge, 1);
  }
```

```
Ran 1 test for test/BridgeFactoryTokenAndLockbox.t.sol:BridgeFactoryTokenAndLockboxTest
[PASS] test_mintVeryHighLimitDOS(address,uint256) (runs: 26, μ: 2698362, ~: 2689943)
Suite result: ok. 1 passed; 0 failed; 0 skipped; finished in 16.67ms (15.41ms CPU time)

Ran 1 test suite in 24.80ms (16.67ms CPU time): 1 tests passed, 0 failed, 0 skipped (1 total tests)
```

BVSS

AO:S/AC:L/AX:L/R:N/S:U/C:N/A:C/I:C/D:C/Y:C (3.5)

Recommendation

It is recommended to add a threshold in the `setBridgeLimits()` function to ensure that no calculated limit will exceed the maximum value of a `uint256` variable, allowing bridges to mint and burn when needed.

Remediation Plan

RISK ACCEPTED: The **Tagus Labs team** made a business decision to accept the risk of this finding and not alter the contracts, stating:
This bug is next to impossible due to the range of values we're working with in this case. In particular, we're dealing with timestamps vs. uint256. It turns into impossible time limits.

References

XERC20.sol#L254

7.4 CENTRALIZATION RISKS DUE TO PRIVILEGED ACCESS BY OWNER

// LOW

Description

Several contracts in scope have owners with privileged rights to perform administrative tasks and need to be trusted to not perform malicious updates.
The administrative functions enable the `owner` to add and remove bridges, adjust their limits, set caps and their durations, designate token destinations, configure contracts, and pause or unpause the bridge.

BVSS

AO:S/AC:L/AX:L/R:N/S:U/C:H/A:H/I:H/D:H/Y:H (3.0)

Recommendation

It is recommended to implement a role-based access control mechanism to allow multiple entities to perform admin tasks and limit powers within the system to users according to their roles, effectively reducing centralization risks.

Remediation Plan

RISK ACCEPTED: The **Tagus Labs team** made a business decision to accept the risk of this finding and not alter the contracts, stating:
We're aware of that, and therefore we use a 3-5 multisig address as the owner.

7.5 POTENTIAL RISKS IN SINGLE-STEP TRANSFER OF OWNERSHIP

// LOW

Description

The current implementation of the `InceptionBridge` contract employs a single-step ownership transfer mechanism, transferring ownership directly to another account in one transaction. This is achieved through the inheritance of OpenZeppelin's `OwnableUpgradeable` contract. Similarly, the `XERC20` contract inherits from OpenZeppelin's `Ownable` contract.
This approach taken for both contracts mentioned does not align with best practices regarding security measures, as the address to which the ownership is transferred should be verified to be active or be willing to act as the owner.

BVSS

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

Recommendation

It is recommended to use OpenZeppelin's `Ownable2Step` or `Ownable2StepUpgradeble` contracts over the `Ownable` and `OwnableUpgradeable` currently in use. Alternatively, consider implementing similar two-step ownership transfer logic into the contract.

Remediation Plan

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

References

XERC20/XERC20.sol#L9
InceptionBridge.sol#L27

7.6 LACK OF CHECK FOR ZERO ADDRESS WHEN ADDING DESTINATION

// INFORMATIONAL

Description

The `_addDestination()` function from the `InceptionBridge` contract has a check to verify that the `_bridgeAddressByChainId` mapping is properly set up and is not equivalent to `address(0`. However, it does not perform the same verification for the `fromToken` or the `toToken` input values.
This may incur in possible denial of service for addresses attempting to execute `deposit()` or `withdraw()`.

BVSS

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

Recommendation

Add a check to ensure that neither the `fromToken` and `toToken` addresses are `address(0)`.

Remediation Plan

SOLVED: The **Tagus Labs team** has addressed the finding in commit `a98d3e2` by following the mentioned recommendation.

Remediation Hash

https://github.com/inceptionlrt/bridge/commit/a98d3e2ab68950083c82f453c6e01e4b1205a2d0

References

InceptionBridgeStorage.sol#L198-L221

7.7 LACK OF CHECK FOR DEPOSITS AND WITHDRAWALS WITH ZERO AMOUNT

// INFORMATIONAL

Description

Throughout the codebase, there are multiple instances where the `deposit` and `withdraw` functions do not check if the amount to deposit or withdraw is zero. This may result in unnecessary gas consumption or unexpected behavior.
The affected functions are:
- `deposit` and `withdraw` functions in the `InceptionBridge` contract.
- `deposit`, `depositTo`, `depositNative`, `depositNativeTo`, `withdraw` and `withdrawTo` functions in the `XERC20Lockbox` contract.

Score

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

Recommendation

Add checks in the aforementioned functions to ensure that the amount to deposit or withdraw is greater than zero.

Remediation Plan

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

References

InceptionBridge.sol
XERC20Lockkbox.sol

7.8 PUBLIC FUNCTIONS NOT CALLED WITHIN THE CONTRACT CAN BE MADE EXTERNAL

// INFORMATIONAL

Description

Throughout the codebase, there are multiple functions that are marked as `public` but are not used internally in the contract they are declared. This may result in unnecessary gas consumption.
The affected functions are:
- `mint`, `burn`, `setLockbox`, `mintingMaxLimitOf`, `burningMaxLimitOf` in the `XERC20` contract.
- `depositNativeTo` in the `XERC20Lockbox` contract.
- `getDeploymentCreate2Address` in the `BridgeFactory` contract.

Score

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

Recommendation

Modify the aforementioned functions with the `external` visibility modifier.

Remediation Plan

SOLVED: The **Tagus Labs team** has addressed the finding in commit `a98d3e2` by following the mentioned recommendation.

Remediation Hash

https://github.com/inceptionlrt/bridge/commit/a98d3e2ab68950083c82f453c6e01e4b1205a2d0

7.9 USE OF POST-INCREMENT OPERATOR

// INFORMATIONAL

Description

In the `deposit` function of the `InceptionBridge` contract, the post-increment operator is used to increment the `_globalNonce` variable:

```
_globalNonce++;
```

This is not the most gas efficient approach to incrementing a variable.

Score

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

Recommendation

Replace the post-increment operator with the pre-increment operator to reduce gas costs. Additionally, the piece of code can be executed inside an `unchecked` block, since realistically it is not possible for the `_globalNonce` to overflow in practice:

```
unchecked{
  ++_globalNonce;
}
```

Remediation Plan

SOLVED: The **Tagus Labs team** has addressed the finding in commit `a98d3e2` by following the mentioned recommendation.

Remediation Hash

https://github.com/inceptionlrt/bridge/commit/a98d3e2ab68950083c82f453c6e01e4b1205a2d0

References

InceptionBridge.sol#L103

7.10 UNUSED IMPORT

// INFORMATIONAL

Description

In the `InceptionBridge.sol` file, OpenZeppelin's `IERC20Metada.sol` is imported but not used in the contract.

Score

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 statement.

Remediation Plan

SOLVED: The **Tagus Labs team** has addressed the finding in commit `a98d3e2` by removing the unused import file.

Remediation Hash

https://github.com/inceptionlrt/bridge/commit/a98d3e2ab68950083c82f453c6e01e4b1205a2d0

References

InceptionBridge.sol#L10

7.11 MISSING ERROR DESCRIPTION

// INFORMATIONAL

Description

The `InitializableTransparentUpgradeableProxy` contract contains `require` statements that are missing error descriptions, particularly in the `initialize()` function:

```
require(_implementation() == address(0));
```

and the `_requireZeroValue()` function:

```
require(msg.value == 0);
```

Score

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

Recommendation

Use descriptive reason strings. Alternatively, it is recommended the use of custom errors to follow the project's error standardization, best practices and to improve gas efficiency.

Remediation Plan

SOLVED: The **Tagus Labs team** has addressed the finding in commit `a98d3e2` by using descriptive reason strings.

Remediation Hash

https://github.com/inceptionlrt/bridge/commit/a98d3e2ab68950083c82f453c6e01e4b1205a2d0

References

InitializableTransparentUpgradeableProxy.sol#L23
InitializableTransparentUpgradeableProxy.sol#L164

7.12 UNLOCKED PRAGMA COMPILERS

// INFORMATIONAL

Description

To ensure stability, it's important that contracts are 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.

Score

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

Recommendation

Lock the pragma version to the same version used during development and testing.

Remediation Plan

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

7.13 PUSH0 IS NOT SUPPORTED BY ALL CHAINS

// INFORMATIONAL

Description

The compiler for Solidity 0.8.20 switches the default target EVM version to Shanghai, which means that the generated bytecode will include PUSH0 opcodes. Be sure to select the appropriate EVM version in case you intend to deploy on a chain apart from mainnet like L2 chains that may not support PUSH0, otherwise deployment of your contracts will fail.

Score

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

Recommendation

Make sure to specify the target EVM version when using Solidity 0.8.20 and above, especially if deploying to L2 chains that may not support the `PUSH0` opcode. Stay informed about the opcode support of different chains to ensure smooth deployment and compatibility.

Remediation Plan

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

7.14 INCEPTIONBRIDGE CONTRACT ASSUMES UNIFORM DECIMAL PLACES ACROSS TOKENS

// INFORMATIONAL

Description

The bridge contract operates under the assumption that all bridgeable tokens across supported chains have uniform decimal places. When tokens are deposited, the system records the quantity of tokens. Conversely, during withdrawals, it is assumed that the exact figure is used to mint tokens for users. If the referenced tokens have different decimal scaling, it can lead to discrepancies between these values. On the other hand, it is understood that since the bridgeable tokens are managed internally, it is expected that they are all issued with the same number of decimal places, which would make it unlikely to impact the current codebase.

Score

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

Recommendation

It is recommended to document this assumption clearly in the system specifications to mitigate any potential errors eventually arising from decimal inconsistencies in future upgrades.

Remediation Plan

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

7.15 USE OF FULL FILE IMPORTS

// INFORMATIONAL

Description

Throughout the codebase, there are multiple instances where full file imports are used. While this does not affect functionality, it is not considered a best practice.

Score

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

Recommendation

Use named import syntax instead of importing full files. This restricts what is being imported to just the named items, not everything in the file, for example:

```
import {MyContract} from "src/MyContract.sol"
```

Remediation Plan

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

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.

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

```
XERC20Lockbox._withdraw(address,uint256) (src/XERC20/XERC20Lockbox.sol#98-105) sends eth to arbitrary user
        Dangerous calls:
        - (_success) = address(_to).call{value: _amount}() (src/XERC20/XERC20Lockbox.sol#101)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#functions-that-send-ether-to-arbitrary-destinations
INFO:Detectors:
BridgeFactory._deployXERC20(string,string) (src/factory/BridgeFactory.sol#82-91) calls abi.encodePacked() with multiple dynamic arguments:
        - _salt = keccak256(bytes)(abi.encodePacked(_name,_symbol,deployer)) (src/factory/BridgeFactory.sol#84)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#abi-encodePacked-collision
INFO:Detectors:
TransparentUpgradeableProxy._fallback() (lib/openzeppelin-contracts/contracts/proxy/transparent/TransparentUpgradeableProxy.sol#84-107) calls Proxy._fallback() (lib/openzeppelin-contracts/contracts/proxy/Proxy.sol#58-61) which halt the execution return(uint256,uint256)(0,returndatasize()()) (lib/openzeppelin-contracts/contracts/proxy/Proxy.sol#42)
TransparentUpgradeableProxy.ifAdmin() (lib/openzeppelin-contracts/contracts/proxy/transparent/TransparentUpgradeableProxy.sol#73-79) calls TransparentUpgradeableProxy._fallback() (lib/openzeppelin-contracts/contracts/proxy/transparent/TransparentUpgradeableProxy.sol#84-107) which halt the execution return(uint256,uint256)(ret + 0x20,mload(uint256)(ret)) (lib/openzeppelin-contracts/contracts/proxy/transparent/TransparentUpgradeableProxy.sol#102)
InitializableTransparentUpgradeableProxy._fallback() (src/proxy/InitializableTransparentUpgradeableProxy.sol#42-65) calls Proxy._fallback() (lib/openzeppelin-contracts/contracts/proxy/Proxy.sol#58-61) which halt the execution return(uint256,uint256)(0,returndatasize()()) (lib/openzeppelin-contracts/contracts/proxy/Proxy.sol#42)
InitializableTransparentUpgradeableProxy.ifAdmin() (src/proxy/InitializableTransparentUpgradeableProxy.sol#31-37) calls InitializableTransparentUpgradeableProxy._fallback() (src/proxy/InitializableTransparentUpgradeableProxy.sol#42-65) which halt the execution return(uint256,uint256)(ret + 0x20,mload(uint256)(ret)) (src/proxy/InitializableTransparentUpgradeableProxy.sol#60)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#incorrect-return-in-assembly
INFO:Detectors:
MultipleDepositor.deposit(address,uint256,address,uint256,uint256) (src/tests/MultipleDepositor.sol#15-31) ignores return value by IERC20(fromToken).transferFrom(msg.sender,address(this),numOfDeposits * amount) (src/tests/MultipleDepositor.sol#22-26)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#unchecked-transfer
INFO:Detectors:
InceptionBridgeStorage.getCurrentStamp(uint256) (src/bridge/InceptionBridgeStorage.sol#174-176) performs a multiplication on the result of a division:
        - (block.timestamp / duration) * duration (src/bridge/InceptionBridgeStorage.sol#175)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#divide-before-multiply
INFO:Detectors:
InceptionBridgeStorage._beforeDeposit() (src/bridge/InceptionBridgeStorage.sol#58-64) uses a dangerous strict equality:
        - _previousSender == tx.origin && _previousDepositBlockNum == block.number (src/bridge/InceptionBridgeStorage.sol#60)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#dangerous-strict-equalities
INFO:Detectors:
InceptionBridgeStorage._beforeDeposit() (src/bridge/InceptionBridgeStorage.sol#58-64) uses tx.origin for authorization: _previousSender == tx.origin && _previousDepositBlockNum == block.number (src/bridge/InceptionBridgeStorage.sol#60)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#dangerous-usage-of-txorigin
INFO:Detectors:
ERC1967Upgrade._upgradeToAndCall(address,bytes,bool) (lib/openzeppelin-contracts/contracts/proxy/ERC1967/ERC1967Upgrade.sol#59-64) ignores return value by Address.functionDelegateCall(newImplementation,data) (lib/openzeppelin-contracts/contracts/proxy/ERC1967/ERC1967Upgrade.sol#62)
ERC1967Upgrade._upgradeBeaconToAndCall(address,bytes,bool) (lib/openzeppelin-contracts/contracts/proxy/ERC1967/ERC1967Upgrade.sol#150-156) ignores return value by Address.functionDelegateCall(IBeacon(newBeacon).implementation(),data) (lib/openzeppelin-contracts/contracts/proxy/ERC1967/ERC1967Upgrade.sol#154)
MultipleDepositor.deposit(address,uint256,address,uint256,uint256) (src/tests/MultipleDepositor.sol#15-31) ignores return value by IERC20(fromToken).approve(address(_bridge),numOfDeposits * amount) (src/tests/MultipleDepositor.sol#27)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#unused-return


INFO:Detectors:
ERC20Permit.constructor(string).name (lib/openzeppelin-contracts/contracts/token/ERC20/extensions/ERC20Permit.sol#44) shadows:
        - ERC20.name() (lib/openzeppelin-contracts/contracts/token/ERC20/ERC20.sol#62-64) (function)
        - IERC20Metadata.name() (lib/openzeppelin-contracts/contracts/token/ERC20/extensions/IERC20Metadata.sol#17) (function)
XERC20.constructor(string,string,address)._name (src/XERC20/XERC20.sol#35) shadows:
        - EIP712._name (lib/openzeppelin-contracts/contracts/utils/cryptography/EIP712.sol#50) (state variable)
        - ERC20._name (lib/openzeppelin-contracts/contracts/token/ERC20/ERC20.sol#45) (state variable)
XERC20.constructor(string,string,address)._symbol (src/XERC20/XERC20.sol#35) shadows:
        - ERC20._symbol (lib/openzeppelin-contracts/contracts/token/ERC20/ERC20.sol#46) (state variable)
InceptionBridge.initialize(address,address).notary (src/bridge/InceptionBridge.sol#36) shadows:
        - InceptionBridgeStorage.notary (src/bridge/InceptionBridgeStorage.sol#14) (state variable)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#local-variable-shadowing
INFO:Detectors:
XERC20.constructor(string,string,address)._factory (src/XERC20/XERC20.sol#35) lacks a zero-check on :
                - FACTORY = _factory (src/XERC20/XERC20.sol#37)
XERC20.setLockbox(address)._lockbox (src/XERC20/XERC20.sol#64) lacks a zero-check on :
                - lockbox = _lockbox (src/XERC20/XERC20.sol#66)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#missing-zero-address-validation
INFO:Detectors:
Modifier TransparentUpgradeableProxy.ifAdmin() (lib/openzeppelin-contracts/contracts/proxy/transparent/TransparentUpgradeableProxy.sol#73-79) does not always execute _; or revertModifier InitializableTransparentUpgradeableProxy.ifAdmin() (src/proxy/InitializableTransparentUpgradeableProxy.sol#31-37) does not always execute _; or revertReference: https://github.com/crytic/slither/wiki/Detector-Documentation#incorrect-modifier
INFO:Detectors:
MultipleDepositor.deposit(address,uint256,address,uint256,uint256) (src/tests/MultipleDepositor.sol#15-31) has external calls inside a loop: _bridge.deposit(fromToken,destinationChain,receiver,amount) (src/tests/MultipleDepositor.sol#29)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation/#calls-inside-a-loop
INFO:Detectors:
Reentrancy in InceptionBridge._deposit(address,uint256,address,uint256) (src/bridge/InceptionBridge.sol#62-85):
        External calls:
        - _safeBurn(fromToken,sender,amount) (src/bridge/InceptionBridge.sol#67)
                - IERC20Mintable(token).burn(account,amount) (src/bridge/InceptionBridge.sol#233)
        - _depositIntoLockbox(lockbox,fromToken,sender,amount) (src/bridge/InceptionBridge.sol#68)
                - returndata = address(token).functionCall(data,SafeERC20: low-level call failed) (lib/openzeppelin-contracts/contracts/token/ERC20/utils/SafeERC20.sol#122)
                - IERC20Mintable(token).burn(account,amount) (src/bridge/InceptionBridge.sol#233)
                - (success,returndata) = target.call{value: value}(data) (lib/openzeppelin-contracts/contracts/utils/Address.sol#135)
                - IERC20(fromToken).safeTransferFrom(sender,address(this),amount) (src/bridge/InceptionBridge.sol#90)
                - IERC20(fromToken).safeApprove(lockbox,amount) (src/bridge/InceptionBridge.sol#91)
                - IXERC20Lockbox(lockbox).deposit(amount) (src/bridge/InceptionBridge.sol#92)
        - metaData = Metadata(Utils.stringToBytes32(IERC20Extra(fromToken).name()),Utils.stringToBytes32(IERC20Extra(fromToken).symbol()),0,address(0)) (src/bridge/InceptionBridge.sol#70)
        External calls sending eth:
        - _depositIntoLockbox(lockbox,fromToken,sender,amount) (src/bridge/InceptionBridge.sol#68)
                - (success,returndata) = target.call{value: value}(data) (lib/openzeppelin-contracts/contracts/utils/Address.sol#135)
        State variables written after the call(s):
        - _globalNonce ++ (src/bridge/InceptionBridge.sol#72)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#reentrancy-vulnerabilities-2
```

```
INFO:Detectors:
Reentrancy in BridgeFactory._deployCreate3(bytes,bytes32) (src/factory/BridgeFactory.sol#116-121):
        External calls:
        - addr = CREATE3.deploy(_salt,bytecode,0) (src/factory/BridgeFactory.sol#117)
        Event emitted after the call(s):
        - ContractCreated(addr) (src/factory/BridgeFactory.sol#119)
Reentrancy in XERC20Lockbox._deposit(address,uint256) (src/XERC20/XERC20Lockbox.sol#112-117):
        External calls:
        - ERC20.safeTransferFrom(msg.sender,address(this),_amount) (src/XERC20/XERC20Lockbox.sol#113)
        - XERC20.mint(_to,_amount) (src/XERC20/XERC20Lockbox.sol#115)
        Event emitted after the call(s):
        - Deposit(_to,_amount) (src/XERC20/XERC20Lockbox.sol#116)
Reentrancy in XERC20Lockbox._withdraw(address,uint256) (src/XERC20/XERC20Lockbox.sol#98-105):
        External calls:
        - XERC20.burn(msg.sender,_amount) (src/XERC20/XERC20Lockbox.sol#99)
        - (_success) = address(_to).call{value: _amount}() (src/XERC20/XERC20Lockbox.sol#101)
        - ERC20.safeTransfer(_to,_amount) (src/XERC20/XERC20Lockbox.sol#103)
        External calls sending eth:
        - (_success) = address(_to).call{value: _amount}() (src/XERC20/XERC20Lockbox.sol#101)
        Event emitted after the call(s):
        - Withdraw(_to,_amount) (src/XERC20/XERC20Lockbox.sol#104)
Reentrancy in BridgeFactory.deployLockbox(address,address,bool) (src/factory/BridgeFactory.sol#68-73):
        External calls:
        - _lockbox = _deployLockbox(_xerc20,_baseToken,_isNative) (src/factory/BridgeFactory.sol#71)
                - _lockbox = CREATE3.deploy(_salt,_bytecode,0) (src/factory/BridgeFactory.sol#107)
                - XERC20(_xerc20).setLockbox(_lockbox) (src/factory/BridgeFactory.sol#109)
                - (success) = proxy.call{value: value}(creationCode) (lib/solmate/src/utils/CREATE3.sol#50)
        External calls sending eth:
        - _lockbox = _deployLockbox(_xerc20,_baseToken,_isNative) (src/factory/BridgeFactory.sol#71)
                - (success) = proxy.call{value: value}(creationCode) (lib/solmate/src/utils/CREATE3.sol#50)
        Event emitted after the call(s):
        - LockboxDeployed(_lockbox) (src/factory/BridgeFactory.sol#72)
Reentrancy in BridgeFactory.deployXERC20(string,string) (src/factory/BridgeFactory.sol#54-57):
        External calls:
        - _xerc20 = _deployXERC20(_name,_symbol) (src/factory/BridgeFactory.sol#55)
                - _xerc20 = CREATE3.deploy(_salt,_bytecode,0) (src/factory/BridgeFactory.sol#88)
                - XERC20(_xerc20).transferOwnership(deployer) (src/factory/BridgeFactory.sol#90)
                - (success) = proxy.call{value: value}(creationCode) (lib/solmate/src/utils/CREATE3.sol#50)
        External calls sending eth:
        - _xerc20 = _deployXERC20(_name,_symbol) (src/factory/BridgeFactory.sol#55)
                - (success) = proxy.call{value: value}(creationCode) (lib/solmate/src/utils/CREATE3.sol#50)
        Event emitted after the call(s):
        - XERC20Deployed(_xerc20) (src/factory/BridgeFactory.sol#56)
Reentrancy in InitializableTransparentUpgradeableProxy.initialize(address,address,bytes) (src/proxy/InitializableTransparentUpgradeableProxy.sol#18-23):
        External calls:
        - _upgradeToAndCall(_logic,_data,false) (src/proxy/InitializableTransparentUpgradeableProxy.sol#20)
                - Address.functionDelegateCall(newImplementation,data) (lib/openzeppelin-contracts/contracts/proxy/ERC1967/ERC1967Upgrade.sol#62)
                - (success,returndata) = target.delegatecall(data) (lib/openzeppelin-contracts/contracts/utils/Address.sol#185)
        Event emitted after the call(s):
        - AdminChanged(_getAdmin(),newAdmin) (lib/openzeppelin-contracts/contracts/proxy/ERC1967/ERC1967Upgrade.sol#115)
                - _changeAdmin(admin_) (src/proxy/InitializableTransparentUpgradeableProxy.sol#21)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#reentrancy-vulnerabilities-3


INFO:Detectors:
ERC20Permit.permit(address,address,uint256,uint256,uint8,bytes32,bytes32) (lib/openzeppelin-contracts/contracts/token/ERC20/extensions/ERC20Permit.sol#49-68) uses timestamp for comparisons
        Dangerous comparisons:
        - require(bool,string)(block.timestamp <= deadline,ERC20Permit: expired deadline) (lib/openzeppelin-contracts/contracts/token/ERC20/extensions/ERC20Permit.sol#58)
XERC20._getCurrentLimit(uint256,uint256,uint256,uint256) (src/XERC20/XERC20.sol#210-226) uses timestamp for comparisons
        Dangerous comparisons:
        - _limit == _maxLimit (src/XERC20/XERC20.sol#217)
        - _timestamp + _DURATION <= block.timestamp (src/XERC20/XERC20.sol#219)
        - _timestamp + _DURATION > block.timestamp (src/XERC20/XERC20.sol#221)
        - _calculatedLimit > _maxLimit (src/XERC20/XERC20.sol#224)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#block-timestamp
INFO:Detectors:
CREATE3.deploy(bytes32,bytes,uint256) (lib/solmate/src/utils/CREATE3.sol#37-52) uses assembly
        - INLINE ASM (lib/solmate/src/utils/CREATE3.sol#42-46)
InceptionBridge.withdraw(bytes,bytes,bytes) (src/bridge/InceptionBridge.sol#104-133) uses assembly
        - INLINE ASM (src/bridge/InceptionBridge.sol#108-111)
        - INLINE ASM (src/bridge/InceptionBridge.sol#127-129)
BridgeFactory._create2(bytes,address) (src/factory/BridgeFactory.sol#25-35) uses assembly
        - INLINE ASM (src/factory/BridgeFactory.sol#28-33)
EthereumVerifier.getMetadata(EthereumVerifier.State) (src/lib/EthereumVerifier.sol#34-40) uses assembly
        - INLINE ASM (src/lib/EthereumVerifier.sol#36-38)
EthereumVerifier._decodeReceiptLogs(EthereumVerifier.State,uint256) (src/lib/EthereumVerifier.sol#75-153) uses assembly
        - INLINE ASM (src/lib/EthereumVerifier.sol#130-132)
        - INLINE ASM (src/lib/EthereumVerifier.sol#142-146)
ProofParser.parseProof(uint256) (src/lib/ProofParser.sol#21-32) uses assembly
        - INLINE ASM (src/lib/ProofParser.sol#24-30)
InitializableTransparentUpgradeableProxy._fallback() (src/proxy/InitializableTransparentUpgradeableProxy.sol#42-65) uses assembly
        - INLINE ASM (src/proxy/InitializableTransparentUpgradeableProxy.sol#59-61)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#assembly-usage
```

```
INFO:Detectors:
Different versions of Solidity are used:
        - Version used: ['>=0.8.0', '^0.8.0', '^0.8.1', '^0.8.2', '^0.8.20', '^0.8.8']
        - >=0.8.0 (lib/solmate/src/utils/Bytes32AddressLib.sol#2)
        - >=0.8.0 (lib/solmate/src/utils/CREATE3.sol#2)
        - ^0.8.0 (lib/openzeppelin-contracts/contracts/access/Ownable.sol#4)
        - ^0.8.0 (lib/openzeppelin-contracts/contracts/interfaces/IERC1967.sol#4)
        - ^0.8.0 (lib/openzeppelin-contracts/contracts/interfaces/IERC20.sol#4)
        - ^0.8.0 (lib/openzeppelin-contracts/contracts/interfaces/IERC20Metadata.sol#4)
        - ^0.8.0 (lib/openzeppelin-contracts/contracts/interfaces/IERC5267.sol#4)
        - ^0.8.0 (lib/openzeppelin-contracts/contracts/interfaces/draft-IERC1822.sol#4)
        - ^0.8.0 (lib/openzeppelin-contracts/contracts/proxy/ERC1967/ERC1967Proxy.sol#4)
        - ^0.8.0 (lib/openzeppelin-contracts/contracts/proxy/Proxy.sol#4)
        - ^0.8.0 (lib/openzeppelin-contracts/contracts/proxy/beacon/IBeacon.sol#4)
        - ^0.8.0 (lib/openzeppelin-contracts/contracts/proxy/transparent/ProxyAdmin.sol#4)
        - ^0.8.0 (lib/openzeppelin-contracts/contracts/proxy/transparent/TransparentUpgradeableProxy.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/ERC20Permit.sol#4)
        - ^0.8.0 (lib/openzeppelin-contracts/contracts/token/ERC20/extensions/IERC20Metadata.sol#4)
        - ^0.8.0 (lib/openzeppelin-contracts/contracts/token/ERC20/extensions/IERC20Permit.sol#4)
        - ^0.8.0 (lib/openzeppelin-contracts/contracts/token/ERC20/utils/SafeERC20.sol#4)
        - ^0.8.0 (lib/openzeppelin-contracts/contracts/utils/Context.sol#4)
        - ^0.8.0 (lib/openzeppelin-contracts/contracts/utils/Counters.sol#4)
        - ^0.8.0 (lib/openzeppelin-contracts/contracts/utils/StorageSlot.sol#5)
        - ^0.8.0 (lib/openzeppelin-contracts/contracts/utils/Strings.sol#4)
        - ^0.8.0 (lib/openzeppelin-contracts/contracts/utils/cryptography/ECDSA.sol#4)
        - ^0.8.0 (lib/openzeppelin-contracts/contracts/utils/math/Math.sol#4)
        - ^0.8.0 (lib/openzeppelin-contracts/contracts/utils/math/SafeCast.sol#5)
        - ^0.8.0 (lib/openzeppelin-contracts/contracts/utils/math/SignedMath.sol#4)
        - ^0.8.1 (lib/openzeppelin-contracts/contracts/utils/Address.sol#4)
        - ^0.8.2 (lib/openzeppelin-contracts/contracts/proxy/ERC1967/ERC1967Upgrade.sol#4)
        - ^0.8.20 (lib/openzeppelin-contracts-upgradeable/contracts/access/OwnableUpgradeable.sol#4)
        - ^0.8.20 (lib/openzeppelin-contracts-upgradeable/contracts/proxy/utils/Initializable.sol#4)
        - ^0.8.20 (lib/openzeppelin-contracts-upgradeable/contracts/utils/ContextUpgradeable.sol#4)
        - ^0.8.20 (lib/openzeppelin-contracts-upgradeable/contracts/utils/PausableUpgradeable.sol#4)
        - ^0.8.20 (lib/openzeppelin-contracts-upgradeable/contracts/utils/ReentrancyGuardUpgradeable.sol#4)
        - ^0.8.20 (src/XERC20/XERC20.sol#2)
        - ^0.8.20 (src/XERC20/XERC20Lockbox.sol#2)
        - ^0.8.20 (src/bridge/InceptionBridge.sol#2)
        - ^0.8.20 (src/bridge/InceptionBridgeStorage.sol#2)
        - ^0.8.20 (src/factory/BridgeFactory.sol#2)
        - ^0.8.20 (src/interfaces/IERC20.sol#2)
        - ^0.8.20 (src/interfaces/IFactory.sol#2)
        - ^0.8.20 (src/interfaces/IInceptionBridge.sol#2)
        - ^0.8.20 (src/interfaces/IInceptionBridgeErrors.sol#2)
        - ^0.8.20 (src/interfaces/IXERC20.sol#2)
        - ^0.8.20 (src/interfaces/IXERC20Lockbox.sol#2)
        - ^0.8.20 (src/lib/CallDataRLPReader.sol#2)
        - ^0.8.20 (src/lib/EthereumVerifier.sol#2)
        - ^0.8.20 (src/lib/ProofParser.sol#2)
        - ^0.8.20 (src/lib/Utils.sol#2)
        - ^0.8.20 (src/proxy/InitializableERC1967Proxy.sol#2)
        - ^0.8.20 (src/proxy/InitializableTransparentUpgradeableProxy.sol#2)
        - ^0.8.20 (src/tests/MultipleDepositor.sol#2)
        - ^0.8.8 (lib/openzeppelin-contracts/contracts/utils/ShortStrings.sol#4)
        - ^0.8.8 (lib/openzeppelin-contracts/contracts/utils/cryptography/EIP712.sol#4)
        - v2 (src/bridge/InceptionBridge.sol#3)
        - v2 (src/bridge/InceptionBridgeStorage.sol#3)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#different-pragma-directives-are-used

INFO:Detectors:
InitializableTransparentUpgradeableProxy._admin() (src/proxy/InitializableTransparentUpgradeableProxy.sol#138-140) is never used and should be removed
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#dead-code
INFO:Detectors:
Low level call in ProxyAdmin.getProxyImplementation(ITransparentUpgradeableProxy) (lib/openzeppelin-contracts/contracts/proxy/transparent/ProxyAdmin.sol#21-27):
        - (success,returndata) = address(proxy).staticcall(0x5c60da1b) (lib/openzeppelin-contracts/contracts/proxy/transparent/ProxyAdmin.sol#24)
Low level call in ProxyAdmin.getProxyAdmin(ITransparentUpgradeableProxy) (lib/openzeppelin-contracts/contracts/proxy/transparent/ProxyAdmin.sol#36-42):
        - (success,returndata) = address(proxy).staticcall(0xf851a440) (lib/openzeppelin-contracts/contracts/proxy/transparent/ProxyAdmin.sol#39)
Low level call in SafeERC20._callOptionalReturnBool(IERC20,bytes) (lib/openzeppelin-contracts/contracts/token/ERC20/utils/SafeERC20.sol#134-142):
        - (success,returndata) = address(token).call(data) (lib/openzeppelin-contracts/contracts/token/ERC20/utils/SafeERC20.sol#139)
Low level call in Address.sendValue(address,uint256) (lib/openzeppelin-contracts/contracts/utils/Address.sol#64-69):
        - (success) = recipient.call{value: amount}() (lib/openzeppelin-contracts/contracts/utils/Address.sol#67)
Low level call in Address.functionCallWithValue(address,bytes,uint256,string) (lib/openzeppelin-contracts/contracts/utils/Address.sol#128-137):
        - (success,returndata) = target.call{value: value}(data) (lib/openzeppelin-contracts/contracts/utils/Address.sol#135)
Low level call in Address.functionStaticCall(address,bytes,string) (lib/openzeppelin-contracts/contracts/utils/Address.sol#155-162):
        - (success,returndata) = target.staticcall(data) (lib/openzeppelin-contracts/contracts/utils/Address.sol#160)
Low level call in Address.functionDelegateCall(address,bytes,string) (lib/openzeppelin-contracts/contracts/utils/Address.sol#180-187):
        - (success,returndata) = target.delegatecall(data) (lib/openzeppelin-contracts/contracts/utils/Address.sol#185)
Low level call in CREATE3.deploy(bytes32,bytes,uint256) (lib/solmate/src/utils/CREATE3.sol#37-52):
        - (success) = proxy.call{value: value}(creationCode) (lib/solmate/src/utils/CREATE3.sol#50)
Low level call in XERC20Lockbox._withdraw(address,uint256) (src/XERC20/XERC20Lockbox.sol#98-105):
        - (_success) = address(_to).call{value: _amount}() (src/XERC20/XERC20Lockbox.sol#101)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#low-level-calls
INFO:Detectors:
Event InceptionBridgeStorage.DEBUG_BYTES(bytes32) (src/bridge/InceptionBridgeStorage.sol#157) is not in CapWords
Function IXERC20Lockbox.XERC20() (src/interfaces/IXERC20Lockbox.sol#35) is not in mixedCase
Function IXERC20Lockbox.ERC20() (src/interfaces/IXERC20Lockbox.sol#37) is not in mixedCase
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#conformance-to-solidity-naming-conventions
INFO:Detectors:
ShortStrings.slitherConstructorConstantVariables() (lib/openzeppelin-contracts/contracts/utils/ShortStrings.sol#40-122) uses literals with too many digits:
        - _FALLBACK_SENTINEL = 0x00000000000000000000000000000000000000000000000000000000000000FF (lib/openzeppelin-contracts/contracts/utils/ShortStrings.sol#42)
BridgeFactory._deployXERC20(string,string) (src/factory/BridgeFactory.sol#82-91) uses literals with too many digits:
        - _creation = type()(XERC20).creationCode (src/factory/BridgeFactory.sol#85)
BridgeFactory._deployLockbox(address,address,bool) (src/factory/BridgeFactory.sol#102-110) uses literals with too many digits:
        - _bytecode = abi.encodePacked(type()(XERC20Lockbox).creationCode,abi.encode(_xerc20,_baseToken,_isNative)) (src/factory/BridgeFactory.sol#105)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#too-many-digits
INFO:Detectors:
CallDataRLPReader.WORD_SIZE (src/lib/CallDataRLPReader.sol#9) is never used in CallDataRLPReader (src/lib/CallDataRLPReader.sol#4-118)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#unused-state-variable
INFO:Detectors:
BridgeFactory.bridgeSalt (src/factory/BridgeFactory.sol#13) should be constant
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#state-variables-that-could-be-declared-constant
INFO:Detectors:
MultipleDepositor._bridge (src/tests/MultipleDepositor.sol#9) should be immutable
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#state-variables-that-could-be-declared-immutable
INFO:Slither:. analyzed (62 contracts with 94 detectors), 58 result(s) found
```

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

UNIT TESTS AND FUZZING

The original repository used the Hardhat environment to develop and test the smart contracts. All tests were executed successfully. Additionally, the project in scope was cloned to a `Foundry` environment, to allow for additional testing and fuzz testing that covered ~50,000 runs per test. These additional tests were run successfully.

```
Ran 34 tests for test/BridgeFactoryTokenAndLockbox.t.sol:BridgeFactoryTokenAndLockboxTest
[PASS] test_bridgeSalt() (gas: 10566)
[PASS] test_burningCurrentLimitOf(address,uint256,uint256) (runs: 50022, μ: 2608015, ~: 2611698)
[PASS] test_burningMaxLimitOf(address,uint256,uint256) (runs: 50022, μ: 2607847, ~: 2611241)
[PASS] test_deployCreate2() (gas: 3277497)
[PASS] test_deployCreate3(bytes32) (runs: 50022, μ: 3299391, ~: 3299391)
[PASS] test_deployLockboxNative() (gas: 2433474)
[PASS] test_deployLockboxNotNative() (gas: 2435706)
[PASS] test_deployLockboxNotTokenOwnerShouldFail(address,bool) (runs: 50022, μ: 1772947, ~: 1772972)
[PASS] test_deployXERC20() (gas: 1767608)
[PASS] test_depositNative(uint256) (runs: 50022, μ: 2494163, ~: 2495987)
[PASS] test_depositNativeTo(address,uint256) (runs: 50021, μ: 2494606, ~: 2496470)
[PASS] test_depositNativeToLockbox(address,uint256) (runs: 50021, μ: 2496879, ~: 2496879)
[PASS] test_depositNativeViaReceive(uint256) (runs: 50022, μ: 2494310, ~: 2496134)
[PASS] test_depositNotNative(address,uint256) (runs: 50021, μ: 2651691, ~: 2651036)
[PASS] test_depositShouldFail(uint256) (runs: 50022, μ: 2434711, ~: 2434711)
[PASS] test_depositToShouldFail(address,uint256) (runs: 50022, μ: 2435101, ~: 2435101)
[PASS] test_depositTokenToLockbox(address,uint256) (runs: 50021, μ: 2648791, ~: 2648082)
[PASS] test_getDeploymentCreate2Address() (gas: 3277454)
[PASS] test_lowLimit(address,uint256,uint256) (runs: 50022, μ: 2606009, ~: 2609677)
[PASS] test_mintMaxLimit(address,uint256,uint256) (runs: 50021, μ: 2662567, ~: 2672236)
[PASS] test_mintVeryHighLimitDOS(address,uint256) (runs: 50021, μ: 2674729, ~: 2684497)
[PASS] test_mintWithEnoughLimit(address,uint256,uint256,uint256) (runs: 50021, μ: 2670044, ~: 2671876)
[PASS] test_mintWithEnoughLimitAfterTime(address,uint256,uint256,uint256) (runs: 50021, μ: 2711548, ~: 2701799)
[PASS] test_mintWithNotEnoughLimitsShouldFail(uint256) (runs: 50022, μ: 1785960, ~: 1785950)
[PASS] test_mintingCurrentLimitOf(address,uint256,uint256) (runs: 50022, μ: 2608032, ~: 2611659)
[PASS] test_mintingMaxLimitOf(address,uint256,uint256) (runs: 50022, μ: 2607644, ~: 2611187)
[PASS] test_renounceOwnership() (gas: 1753797)
[PASS] test_setBridgeLimits(address,uint256,uint256) (runs: 50022, μ: 2605958, ~: 2609617)
[PASS] test_setBridgeLimitsByAnyUserShouldFail(address,uint256,uint256) (runs: 50022, μ: 2437807, ~: 2437807)
[PASS] test_transferOwnership(address) (runs: 50021, μ: 1774483, ~: 1774483)
[PASS] test_withdraw(uint256,uint256) (runs: 50021, μ: 2534593, ~: 2538751)
[PASS] test_withdrawTo(uint256,uint256,address) (runs: 50020, μ: 2542074, ~: 2546241)
[PASS] test_withdrawToLockboxNative(uint256) (runs: 50022, μ: 2510459, ~: 2510448)
[PASS] test_withdrawToLockboxNotNative(uint256,uint256) (runs: 50022, μ: 2767110, ~: 2768979)
Suite result: ok. 34 passed; 0 failed; 0 skipped; finished in 323.81s (2886.96s CPU time)

Ran 30 tests for test/InceptionBridge.t.sol:InceptionBridgeTest
[PASS] test_addBridge(address,uint256) (runs: 50022, μ: 45716, ~: 46722)
[PASS] test_addDestination(address,address,uint256,address) (runs: 50020, μ: 81019, ~: 81019)
[PASS] test_addDestinationToTokenZeroAddress(address,address,uint256,address) (runs: 50020, μ: 57722, ~: 57722)
[PASS] test_deployLockboxNativeBridge() (gas: 5566337)
[PASS] test_depositBridgeWhenLockboxAddressZero(address,address,uint256,address,uint256,uint256,uint256) (runs: 50020, μ: 315488, ~: 315418)
[PASS] test_depositBridgeWhenLockboxIsNotAddressZero(address,address,uint256,address,uint256,uint256,uint256,uint256,uint256) (runs: 50020, μ: 604003, ~: 607281)
[PASS] test_getCurrentStamp(uint256) (runs: 50022, μ: 20009, ~: 19796)
[PASS] test_getDestination(address,address,uint256,address) (runs: 50020, μ: 89008, ~: 89008)
[PASS] test_initialize(address,address) (runs: 50022, μ: 21060, ~: 21060)
[PASS] test_longCapDuration(uint256) (runs: 50022, μ: 31564, ~: 31755)
[PASS] test_longCaps(address,uint256) (runs: 50022, μ: 45904, ~: 46705)
[PASS] test_notary(address) (runs: 50021, μ: 37587, ~: 37588)
[PASS] test_owner() (gas: 20020)
[PASS] test_pause() (gas: 47488)
[PASS] test_removeBridge(address,uint256) (runs: 50021, μ: 38649, ~: 38658)
[PASS] test_removeDestination(address,address,uint256,address) (runs: 50020, μ: 70454, ~: 70454)
[PASS] test_renounceOwnership() (gas: 24121)
[PASS] test_setLongCap(address,uint256) (runs: 50022, μ: 49950, ~: 50703)
[PASS] test_setLongCapDuration(uint256) (runs: 50022, μ: 31516, ~: 31711)
[PASS] test_setNotary(address) (runs: 50022, μ: 30808, ~: 30810)
[PASS] test_setShortCap(address,uint256) (runs: 50022, μ: 49758, ~: 50570)
[PASS] test_setShortCapDuration(uint256) (runs: 50022, μ: 31581, ~: 31778)
[PASS] test_setXERC20Lockbox(address,address) (runs: 50022, μ: 44550, ~: 44558)
[PASS] test_shortCapDuration(uint256) (runs: 50022, μ: 31605, ~: 31798)
[PASS] test_shortCaps(address,uint256) (runs: 50022, μ: 45758, ~: 46550)
[PASS] test_transferOwnership(address) (runs: 50021, μ: 29683, ~: 29684)
[PASS] test_unpause() (gas: 39674)
[PASS] test_withdrawBridgeLockboxNotSet() (gas: 330576)
[PASS] test_withdrawBridgeLockboxSet() (gas: 678391)
[PASS] test_xerc20TokenRegistry(address,address) (runs: 50021, μ: 47437, ~: 47437)
Suite result: ok. 30 passed; 0 failed; 0 skipped; finished in 323.81s (433.45s CPU time)

Ran 2 test suites in 323.82s (647.62s CPU time): 64 tests passed, 0 failed, 0 skipped (64 total tests)
```

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.