Zellic

Scroll Tech

Smart Contract Security Assessment

May 26, 2023

Prepared for:

Haichen Shen

Scroll

Prepared by:

Ayaz Mammadov and Vlad Toie

Zellic Inc.

Contents

About Zellic

Zellic was founded in 2020 by a team of blockchain specialists with more than a decade of combined industry experience. We are leading experts in smart contracts and Web3 development, cryptography, web security, and reverse engineering. Before Zellic, we founded perfect blue, the top competitive hacking team in the world. Since then, our team has won countless cybersecurity contests and blockchain security events.

Zellic aims to treat clients on a case-by-case basis and to consider their individual, unique concerns and business needs. Our goal is to see the long-term success of our partners rather than simply provide a list of present security issues. Similarly, we strive to adapt to our partners' timelines and to be as available as possible. To keep up with our latest endeavors and research, check out our website zellic.io or follow @zellic_io on Twitter. If you are interested in partnering with Zellic, please contact us at hello@zellic.io.

1 Executive Summary

Zellic conducted a security assessment for Scroll from March 20th to April 3rd, 2023. During this engagement, Zellic reviewed Scroll Tech's code for security vulnerabilities, design issues, and general weaknesses in security posture.

1.1 Goals of the Assessment

In a security assessment, goals are framed in terms of questions that we wish to answer. These questions are agreed upon through close communication between Zellic and the client. In this assessment, we sought to answer the following questions:

- How does Scroll validate the transfers of tokens between the two layers?
- Are there any potential exploits in the design of each Messenger contract that could result in fund theft?
- Could there be any vulnerabilities in the verification of the proofs generated by the off-chain proof generation process?
- Can fake tokens be supplied for legitimate other-layer tokens?

1.2 Non-goals and Limitations

We did not assess the following areas that were outside the scope of this engagement:

- Front-end components
- Infrastructure relating to the project, such as off-chain proof generation
- Key custody

Due to the time-boxed nature of security assessments in general, there are limitations in the coverage an assessment can provide.

During this assessment, the length and complexity of the verifier functions from `ScrollChainCommitmentVerifier`,`PatriciaMerkleTrieVerifier` prevented us from performing a in-depth verification of their correctness. We have therefore only performed a high-level review of these functions.

1.3 Results

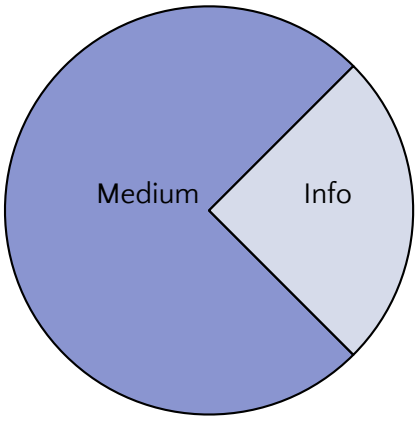
During our assessment on the scoped Scroll Tech contracts, we discovered four findings. No critical issues were found. Of the four findings, three were of medium impact,

and the remaining findings were informational in nature.

Additionally, Zellic recorded its notes and observations from the assessment for Scroll's benefit in the Discussion section (4) at the end of the document.

Breakdown of Finding Impacts

Impact Level	Count
Critical	0
High	0
Medium	3
Low	0
Informational	1



2 Introduction

2.1 About Scroll Tech

Scroll is a zkEVM-based zkRollup on Ethereum that enables native compatibility for existing Ethereum applications and tools.

2.2 Methodology

During a security assessment, Zellic works through standard phases of security auditing including both automated testing and manual review. These processes can vary significantly per engagement, but the majority of the time is spent on a thorough manual review of the entire scope.

Alongside a variety of tools and analyzers used on an as-needed basis, Zellic focuses primarily on the following classes of security and reliability issues:

Basic coding mistakes. Many critical vulnerabilities in the past have been caused by simple, surface-level mistakes that could have easily been caught ahead of time by code review. Depending on the engagement, we may also employ sophisticated analyzers such as model checkers, theorem provers, fuzzers, and so on as necessary. We also perform a cursory review of the code to familiarize ourselves with the contracts.

Business logic errors. Business logic is the heart of any smart contract application. We examine the specifications and designs for inconsistencies, flaws, and weaknesses that create opportunities for abuse. For example, these include problems like unrealistic tokenomics or dangerous arbitrage opportunities. To the best of our abilities, time permitting, we also review the contract logic to ensure that the code implements the expected functionality as specified in the platform's design documents.

Integration risks. Several well-known exploits have not been the result of any bug within the contract itself; rather, they are an unintended consequence of the contract's interaction with the broader DeFi ecosystem. Time permitting, we review the contracts' external interactions and summarize the associated risks: for example, flash loan attacks, oracle price manipulation, MEV/sandwich attacks, and so on.

Code maturity. We look for potential improvements in the code base in general. We look for violations of industry best practices and guidelines and code quality standards. We also provide suggestions for possible optimizations, such as gas optimization, upgradeability weaknesses, centralization risks, and so on.

For each finding, Zellic assigns it an impact rating based on its severity and likelihood.

There is no hard-and-fast formula for calculating a finding's impact. Instead, we assign it on a case-by-case basis based on our judgment and experience. Both the severity and likelihood of an issue affect its impact. For instance, a highly severe issue's impact may be attenuated by a low likelihood. We assign the following impact ratings (ordered by importance): Critical, High, Medium, Low, and Informational.

Zellic organizes its reports such that the most important findings come first in the document, rather than being strictly ordered on impact alone. Thus, we may sometimes emphasize an "Informational" finding higher than a "Low" finding. The key distinction is that although certain findings may have the same impact rating, their *importance* may differ. This varies based on various soft factors, like our clients' threat models, their business needs, and so on. We aim to provide useful and actionable advice to our partners considering their long-term goals, rather than a simple list of security issues at present.

2.3 Scope

The engagement involved a review of the following targets:

Scroll Tech Contracts

Repository https://github.com/scroll-tech/scroll

Version scroll: `cb6a609366e5f6b808dfd9bba01a1c7c6e07f6fa`

Programs

- External.sol
- L1ScrollMessenger.sol
- L1CustomERC20Gateway.sol
- L1ERC1155Gateway.sol
- L1ERC20Gateway.sol
- L1ERC721Gateway.sol
- L1ETHGateway.sol
- L1GatewayRouter.sol
- L1StandardERC20Gateway.sol
- L1WETHGateway.sol
- L1MessageQueue.sol

- L2GasPriceOracle.sol
- ScrollChainCommitmentVerifier.sol
- ScrollChain.sol
- L2ScrollMessenger.sol
- L2CustomERC20Gateway.sol
- L2ERC1155Gateway.sol
- L2ERC20Gateway.sol
- L2ERC721Gateway.sol
- L2ETHGateway.sol
- L2GatewayRouter.sol
- L2StandardERC20Gateway.sol
- L2WETHGateway.sol
- L1BlockContainer.sol
- L1GasPriceOracle.sol
- L2MessageQueue.sol
- WETH9.sol
- ScrollMessengerBase.sol
- AddressAliasHelper.sol
- AppendOnlyMerkleTree.sol
- OwnableBase.sol
- ScrollConstants.sol
- ScrollPredeploy.sol
- ScrollGatewayBase.sol
- SimpleGasOracle.sol
- ScrollStandardERC20Factory.sol
- ScrollStandardERC20.sol
- PatriciaMerkleTrieVerifier.sol

- RollupVerifier.sol
- WithdrawTrieVerifier.sol
- ZkTrieVerifier.sol
- MockERC20.sol
- MockPatriciaMerkleTrieVerifier.sol
- MockZkTrieVerifier.sol
- MockScrollChain.sol
- MockScrollMessenger.sol
- FeeOnTransferToken.sol
- TransferReentrantToken.sol

Type Solidity

Platform EVM-compatible

2.4 Project Overview

Zellic was contracted to perform a security assessment with two consultants for a total of four person-weeks. The assessment was conducted over the course of two calendar weeks.

Contact Information

The following project managers were associated with the engagement:

Chad McDonald, Engagement Manager
chad@zellic.io

The following consultants were engaged to conduct the assessment:

Ayaz Mammadov, Engineer
ayaz@zellic.io

Vlad Toie, Engineer
vlad@zellic.io

2.5 Project Timeline

The key dates of the engagement are detailed below.

March 20, 2023	Kick-off call
March 20, 2023	Start of primary review period
April 3, 2023	End of primary review period
May 25, 2023	Closing call

3 Detailed Findings

3.1 Gateways call the Scroll bridge without supplying a fee

- **Target**: L2ERC721Gateway, L2ERC1155Gateway
- **Category**: Business Logic
- **Severity**: Medium
- **Likelihood**: High
- **Impact**: Medium

Description

The L2ERC721Gateway and L2ERC1155Gateway contracts perform cross-chain invocations by calling the `sendMessage` function in several different functions. However, these contracts do not send any native value or send an equal amount of native tokens and specify the amount to be the same. This results in no fee being left for the bridge, causing the call to always revert.

An example from L2ERC721Gateway can be found below.

```
function _withdrawERC721(
    address _token,
    address _to,
    uint256 _tokenId,
    uint256 _gasLimit
) internal nonReentrant {
    ...
    IL2ScrollMessenger(messenger).sendMessage(counterpart, msg.value,
    _message, _gasLimit);
    ...
}
```

Impact

The gateways are not functional, and the cross-chain invocations made by the L2ERC721Gateway and the L2ERC1155Gateway will always fail and revert.

Recommendations

Change the business logic to account for the bridge fee.

Remediation

This issue has been acknowledged by Scroll, and a fix was implemented in commit 7fb4d1d3.

3.2 ERC1155 token minting may fail

- **Target**: L2ERC1155Gateway
- **Category**: Business Logic
- **Likelihood**: Medium
- **Severity**: Medium
- **Impact**: Medium

Description

When an ERC1155 token is transferred from L1ERC1155Gateway to the L2ERC1155Gateway, the `finalizeDepositERC1155` or `finalizeBatchDepositERC1155` functions are called. These, in turn, call the `mint` or `batchMint` functions of the underlying ERC1155 contract.

A particular detail of both of these functions is the fact that upon minting with `mint`, a callback is triggered on the destination address.

```
function finalizeDepositERC1155(
    address _l1Token,
    address _l2Token,
    address _from,
    address _to,
    uint256 _tokenId,
    uint256 _amount
) external override nonReentrant onlyCallByCounterpart {
    IScrollERC1155(_l2Token).mint(_to, _tokenId, _amount, "");

    emit FinalizeDepositERC1155(_l1Token, _l2Token, _from, _to, _tokenId,
    _amount);
}
```

Here is the code snippet with the callback from the original ERC1155 contract:

```
function _mint(address to, uint256 id, uint256 amount, bytes memory data)
    internal virtual {
    require(to ≠ address(0), "ERC1155: mint to the zero address");

    address operator = _msgSender();
    uint256[] memory ids = _asSingletonArray(id);
    uint256[] memory amounts = _asSingletonArray(amount);

    _beforeTokenTransfer(operator, address(0), to, ids, amounts, data);
```

```
        _balances[id][to] += amount;
        emit TransferSingle(operator, address(0), to, id, amount);

        _afterTokenTransfer(operator, address(0), to, ids, amounts, data);

        _doSafeTransferAcceptanceCheck(operator, address(0), to, id, amount,
        data);
    }
```

The `_doSafeTransferAcceptanceCheck` function is responsible for triggering the callback function `onERC1155Received` if the `_to` address is a contract. However, if the contract at `_to` does not implement the `IERC721Receiver` interface, the `onERC1155Received` function will not be called, resulting in the failure of the `mint` function.

Impact

Should `_to` be a contract that does not inherit the `IERC1155Receiver` interface, the `mint` function will fail, leading to the token not being minted on the L2 side as well as locking the funds on the L1 side. To solve this issue, the protocol will require the manual intervention of the `counterpart` address to mint the token on the L2 side to another `_to` address, possibly escalating into a dispute if the funds are not promptly released.

Recommendations

We recommend exercising additional caution in this scenario because a reversion could result in the ERC1155 becoming stuck in the L1 gateway. Ideally, in the future, a system should be implemented to check the Merkle tree and recover the ERC1155 if the message fails on L2.

Remediation

This issue has been acknowledged by Scroll.

3.3 Arbitrary calldata calls on arbitrary addresses

- **Target**: L1ScrollMessenger, L2ScrollMessenger, L1ETHGateway, L2ETHGateway
- **Category**: Business Logic
- **Severity**: Medium
- **Likelihood**: N/A
- **Impact**: Medium

Description

The messenger contracts allow for the relaying of messages from one chain to the other. As currently implemented, they perform a low-level call to an arbitrary address with arbitrary calldata, both supplied as messages over the bridge. This allows for the execution of external calls in the context of the messenger contract, which essentially means that calls are executed on behalf of the messenger contract.

```
function relayMessageWithProof(
    address _from,
    address _to,
    uint256 _value,
    uint256 _nonce,
    bytes memory _message,
    L2MessageProof memory _proof
) external override whenNotPaused onlyWhitelistedSender(msg.sender) {

    // ...

    // @todo check more `_to` address to avoid attack.
    require(_to ≠ messageQueue, "Forbid to call message queue");
    require(_to ≠ address(this), "Forbid to call self");

    // @note This usually will never happen, just in case.
    require(_from ≠ xDomainMessageSender, "Invalid message sender");

    xDomainMessageSender = _from;
    (bool success, ) = _to.call{value: _value}(_message);
```

Currently, the `_to` is checked against the message queue and the messenger contract itself. However, it is not checked against any other contracts, so theoretically it can call any contract's functions.

Similarly, the gateways allow moving native funds from one chain to the other. The same low-level call is used; however, the calldata is currently not passed over the

bridge. In the future, however, it could be passed over the bridge, allowing for the execution of arbitrary external calls.

```
function finalizeWithdrawETH(
    address _from,
    address _to,
    uint256 _amount,
    bytes calldata _data
) external payable override onlyCallByCounterpart {

    // @note can possible trigger reentrant call to this contract or
    messenger,
    // but it seems not a big problem.
    // solhint-disable-next-line avoid-low-level-calls
    (bool _success, ) = _to.call{value: _amount}("");
    require(_success, "ETH transfer failed");

    // @todo farward _data to `_to` in near future.

    emit FinalizeWithdrawETH(_from, _to, _amount, _data);
}
```

Impact

Due to the nature of arbitrary calls, it is likely that an attack could directly steal any type of funds (ERC20, ERC721, native, etc.) from the messenger contracts. Moreover, should users give allowance to the messenger contracts, the attacker could also steal any ERC20 tokens that the user has given allowance for, by calling the `transferFrom` function on the respective ERC20 contracts.

At present, there is no immediate security concern for this finding. However, it is worth noting that if data forwarding components are naively implemented then it opens up an avenue for a critical bug, the details follow:

In the case of the gateways, an attacker could supply the L2ScrollMessenger itself as the target and supply data using the data forwarding feature such that the L1ETHGateway could be tricked into giving away ETH, as any message could be forged on behalf of the L2ETHGateway and it would be considered legitimate on the L1 side because it came from the appropriate counterpart.

The total attack chain would look like this:

```
L1EthGateway → L1ScrollMessenger → L2ScrollMessenger → L2EthGateway
    → finalizeDepositETH → _to.call → L2ScrollMessenger
    → L1ScrollMessenger → L1GatewayETH (Withdraw any amount of ETH)
```

Another consequence of these direct calls is that user errors, such as providing approval incorrectly to the scroll messenger, can be exploited by malicious users. They can make cross-chain calls from one side of the chain to the other, supplying call data to specific smart contracts or tokens, to execute functions like `transferFrom`.

Recommendations

We recommend ensuring that the `_to` address is a contract and that it implements a custom interface. This way, even if the contract is an arbitrary one, it will need to follow Scroll's interface, ensuring the context of the call is correct and no arbitrary actions can be performed on behalf of the messenger or gateway contracts.

An example of such an interface could be

```
// SPDX-License-Identifier: MIT

pragma solidity ^0.8.0;

interface IScrollCallback {
    /// @notice Handle the callback from L1/L2 contracts.
    /// @param to The address of recipient's account on L1/L2.
    /// @param amount The amount of ETH to be deposited.
    function handleContractCallback(
        bytes memory message
    ) external payable;
}
```

The messenger contract would then check that the `_to` contract implements the interface and call it with the message as the argument.

```
function relayMessageWithProof(
    address _from,
    address _to,
    uint256 _value,
    uint256 _nonce,
    bytes memory _message,
```

```
    L2MessageProof memory _proof
) external override whenNotPaused onlyWhitelistedSender(msg.sender) {

    // ...

    // @todo check more `_to` address to avoid attack.
    require(_to ≠ messageQueue, "Forbid to call message queue");
    require(_to ≠ address(this), "Forbid to call self");

    // @note This usually will never happen, just in case.
    require(_from ≠ xDomainMessageSender, "Invalid message sender");

    xDomainMessageSender = _from;
    (bool success, ) = _to.call{value: _value}(_message);`

    bytes memory payload = abi.encodeWithSelector(
        IScrollCallback.handleContractCallback.selector,
        _message
    );
    (bool success, ) = _to.call{value: _value}(payload);
}
```

Remediation

The issue has been acknowledged by Scroll, and a fix was implemented in commit bfe29b41. It's important to note that neither the `L1ScrollMessenger` nor the `L2ScrollMessenger` have been updated with the fix, as the Scroll team is yet to decide on the best way to implement it.

3.4 Underlying initializers are not called

- **Target**: L1ERC721Gateway, L2ERC721Gateway, L1ERC1155Gateway, L2ERC1155Gateway, ScrollStandardERC20
- **Category**: Coding Mistakes
- **Severity**: Low
- **Likelihood**: N/A
- **Impact**: Informational

Description

The initialize functions from the contracts listed above fail to call all the internal initializers of the contracts they inherit from. This may result in unforeseen consequences if the parent contracts are modified later on, as some state variables may not be properly initialized.

Impact

Currently, the missing initializers do not contain any important logic. Hence, there are no security implications.

Recommendations

We recommend that the initializers of the contracts listed above are updated to call the underlying initializers of the contracts they inherit from. This can be done by adding the following lines to each of the affected `initialize` functions:

In L1ERC721Gateway and L2ERC721Gateway,

```
function initialize(address _counterpart, address _messenger)
    external initializer {
    OwnableUpgradeable.__Ownable_init();
    ScrollGatewayBase._initialize(_counterpart, address(0), _messenger);
    ERC721HolderUpgradeable.__ERC721Holder_init();
}
```

In L1ERC1155Gateway and L2ERC1155Gateway,

```
function initialize(address _counterpart, address _messenger)
    external initializer {
    OwnableUpgradeable.__Ownable_init();
    ScrollGatewayBase._initialize(_counterpart, address(0), _messenger);
    ERC1155HolderUpgradeable.__ERC1155Holder_init();
```

```
}
```

In ScrollStandardERC20,

```
function initialize(
    string memory _name,
    string memory _symbol,
    uint8 _decimals,
    address _gateway,
    address _counterpart
) external initializer {
    __ERC20Permit_init(_name);
    __ERC20_init(_name, _symbol);
    ContextUpgradeable.__Context_init();
    decimals_ = _decimals;
    gateway = _gateway;
    counterpart = _counterpart;
}
```

Remediation

This issue has been acknowledged by Scroll, and a fix was implemented in commit 2728cfa9.

4 Discussion

The purpose of this section is to document miscellaneous observations that we made during the assessment.

4.1 Additional assurance checks

Assurance checks are an important part of the security of the system. They are not a replacement for the security of the contracts themselves, but they are a way to ensure that the contracts are being used as intended.

Throughout the audit, we have identified a number of checks that should be added to the contracts to ensure that they are being used as intended. Here we list them and explain why they are important.

L2StandardERC20Gateway

The `finalizeDepositERC20` function in L2StandardERC20Gateway should check that the `deployData` is not empty, such that the newly deployed token is constructed properly.

```
function finalizeDepositERC20(
    address _l1Token,
    address _l2Token,
    address _from,
    address _to,
    uint256 _amount,
    bytes calldata _data
) external payable override onlyCallByCounterpart {
    // ...

    if (tokenMapping[_l2Token] == address(0)) {
        // first deposit, update mapping
        tokenMapping[_l2Token] = _l1Token;
        (_callData, _deployData) = abi.decode(_data, (bytes, bytes));
    } else {

        // @audit should it be assured that tokenMapping[_l2Token] ==
    _l1Token?
        _callData = _data;
```

```
    }

    require(_deployData.length > 0);

    if (!_l2Token.isContract()) {
        _deployL2Token(_deployData, _l1Token);
    }

    // ...
}
```

ScrollStandardERC20Factory

The `deployL2Token` deterministically deploys a new token on L2 based on a computed salt. However, it does not check that the token has not already been deployed. We recommend adding a check to ensure that the token has not already been deployed.

```
function deployL2Token(address _gateway, address _l1Token)
    external onlyOwner returns (address) {
    bytes32 _salt = _getSalt(_gateway, _l1Token);

    require(hasBeenDeployed[_salt] == false);

    return Clones.cloneDeterministic(implementation, _salt);
}
```

No fake tokens should be supplied

At a project-wide level, checks should be performed such that no fake tokens can be supplied. This is important because the system relies on the fact that the tokens are the same on both sides of the bridge. If a fake token is supplied on one side for a legitimate token on the other side, the system should not allow the transfer to happen. Additionally, extra caution should be taken when updating the token mapping, such that no two different tokens can point to the same L1 or L2 token.

We have identified a number of places where this check, or additional checks, should be added:

- In L2StandardERC20Gateway.sol for the `finalizeDepositERC20` function,

```
function finalizeDepositERC20(
    address _l1Token,
    address _l2Token,
    address _from,
    address _to,
    uint256 _amount,
    bytes calldata _data
) external payable override onlyCallByCounterpart {
    // ...

    if (tokenMapping[_l2Token] == address(0)) {
        // first deposit, update mapping
        tokenMapping[_l2Token] = _l1Token;
        (_callData, _deployData) = abi.decode(_data, (bytes, bytes));
    } else {

        require(tokenMapping[_l2Token] == _l1Token);
        require(_l1Token ≠ address(0));
        _callData = _data;
    }
```

- In L1CustomERC20Gateway.sol for the `finalizeWithdrawERC20` function - similarly for L1ERC1155Gateway.sol and L1ERC721Gateway.sol,

```
function finalizeWithdrawERC20(
    address _l1Token,
    address _l2Token,
    address _from,
    address _to,
    uint256 _amount,
    bytes calldata _data
) external payable override onlyCallByCounterpart {
    // ...

    require(msg.value == 0, "nonzero msg.value");
    require(_l2Token == tokenMapping[_l1Token], "l2 token mismatch");
    require(_l1Token ≠ address(0));

    // ...
}
```

- In L1CustomERC20Gateway.sol for the `updateTokenMapping` function - similarly for L1StandardERC20Gateway, L1ERC1155Gateway.sol and L1ERC721Gateway.sol,

```
function finalizeWithdrawERC20(
    address _l1Token,
    address _l2Token,
    address _from,
    address _to,
    uint256 _amount,
    bytes calldata _data
) external payable override onlyCallByCounterpart {
    require(msg.value == 0, "nonzero msg.value");
    require(_l1Token ≠ address(0));
    require(tokenMapping[_l1Token] == _l2Token, "TM already set");
```

- In L2ERC721Gateway.sol for the `finalizeDepositERC721` and `finalizeBatchDepositERC721` functions - similarly for L2CustomERC20Gateway in `finalizeDepositERC20` and L2ERC1155Gateway.sol with the `finalizeDepositERC1155` and `finalizeBatchDepositERC1155` functions,

```
function finalizeDepositERC721(
    address _l1Token,
    address _l2Token,
    address _from,
    address _to,
    uint256 _tokenId
) external override nonReentrant onlyCallByCounterpart {
    require(tokenMapping[_l2Token] == _l1Token);
```

- Moreover, in every `updateTokenMapping` function, checks should be performed such that the token mapping is not already set, to prevent overwriting it:

```
function updateTokenMapping(address _l1Token, address _l2Token)
    external onlyOwner {
    require(_l2Token ≠ address(0), "map to zero address");
    require(tokenMapping[_l1Token] == address(0), "TM already set");

    tokenMapping[_l1Token] = _l2Token;
```

```
        emit UpdateTokenMapping(_l1Token, _l2Token);
    }
```

4.2 Missing `msg.value` checks

The L1ETHGateway and L2ETHGateway contracts handle the retrieval of ETH on the L1 and L2 chains, respectively.

Currently, the functions used to finalize cross-chain transfers are `finalizeDepositETH` in L2ETHGateway.sol and `finalizeWithdrawETH` in L1ETHGateway.sol. These functions are only callable by the counterpart; however, they do not check that the `msg.value` is equal to the `amount` of ETH being transferred.

This could be used to trick the gateway into thinking that more ETH was deposited than was actually supposed to be deposited or withdrawn. Currently, since it is an `onlyCallByCounterpart` function, it is not possible to call it directly, unless private keys are compromised.

We recommend adding a check to ensure that the `msg.value` is equal to the `amount` of ETH being transferred.

```
function finalizeDepositETH(
    address _from,
    address _to,
    uint256 _amount,
    bytes calldata _data
) external payable override onlyCallByCounterpart {
    // ...
    require(msg.value == _amount);

    // ...
}

function finalizeWithdrawETH(
    address _from,
    address _to,
    uint256 _amount,
    bytes calldata _data
) external payable override onlyCallByCounterpart {
    // ...
```

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

    // ...
}
```

4.3 To-dos

During this audit, we have identified a significant amount of code in the codebase that is currently a work in progress or yet to be completed. It is important to note that the changes that will follow are not covered in the scope of this audit. There are certain areas that are noteworthy to mention:

- The replay system on the L1 system is currently not implemented.
- The relay system is currently centralized as only whitelisted `msg.senders` can relay a message.
- Many data-forwarding features in gateways are not implemented.

5 Threat Model

This provides a full threat model description for various functions. As time permitted, we analyzed each function in the smart contracts and created a written threat model for some critical functions. A threat model documents a given function's externally controllable inputs and how an attacker could leverage each input to cause harm.

Not all functions in the audit scope may have been modeled. The absence of a threat model in this section does not necessarily suggest that a function is safe.

5.1 Module: L1CustomERC20Gateway.sol

Function: `finalizeWithdrawERC20(address _l1Token, address _l2Token, address _from, address _to, uint256 _amount, byte[] _data)`

Perform the finalization of withdraw ERC20 token from layer 2 to layer 1.

Inputs

- `_l1Token`
  - **Control**: Full control.
  - **Constraints**: Checked `tokenMapping[_l1Token] == _l2Token`.
  - **Impact**: l1Token to be withdrawn.
- `_l2Token`
  - **Control**: Full control.
  - **Constraints**: Checked `tokenMapping[_l1Token] == _l2Token`.
  - **Impact**: l2Token that was burned.
- `_from`
  - **Control**: Full control.
  - **Constraints**: N/A (checked in l2).
  - **Impact**: Sender from l2.
- `_to`
  - **Control**: Full control.
  - **Constraints**: N/A.
  - **Impact**: Destination address.
- `_amount`
  - **Control**: Full control.
  - **Constraints**: N/A (checked in l2).

  - **Impact**: Amount to be withdrawn.
- `_data`
  - **Control**: Full control.
  - **Constraints**: N/A (checked in l2).
  - **Impact**: Data to be passed on to the recipient.

Branches and code coverage (including function calls)

Intended branches

- Should ensure that `_l2Token == tokenMapping[_l1Token] && l2Token ≠ address(0)`.
  ☐ Test coverage
- Increase the balance of `_l1Token` by `_amount` for the `_to`.
  ☑ Test coverage
- Decrease the balance of `_l1Token` by `_amount` for the `address(this)`.
  ☑ Test coverage
- Assume the `counterpart` is legitimate and only calls this function when an identical `amount` of `_l2Token` is burned on layer 2.
  ☑ Test coverage

Negative behavior

- Should not be callable by anyone other than the counterpart.
  ☑ Negative test
- Should not be callable multiple times for the same burn.
  ☐ Negative test
- Should not be callable with a zero `_amount`.
  ☑ Negative test

Function call analysis

- `IERC20Upgradeable(_l1Token).safeTransfer(_to, _amount)`
  - **What is controllable?** `_l1Token`.
  - **If return value controllable, how is it used and how can it go wrong?** N/A; assumed that the `l1token` has been previously validated.
  - **What happens if it reverts, reenters, or does other unusual control flow?** Transfer fails and the function reverts

Function: `updateTokenMapping(address _l1Token, address _l2Token)`

Update the mapping of layer 1 to layer 2 token.

Inputs

- `_l1Token`
  - **Control**: Full control.
  - **Constraints**: N/A.
  - **Impact**: The layer 1 token to be approved.
- `_l2Token`
  - **Control**: Full control.
  - **Constraints**: Checked that `_l2Token` ≠ `address(0)`.
  - **Impact**: The layer 2 token to be approved.

Branches and code coverage (including function calls)

Intended branches

- Assumed this should not change that often, or at least not during important bridge operations.
  - ☐ Test coverage
- Update the state of the `tokenMapping` to reflect the addition.
  - ☑ Test coverage

Negative behavior

- Should not be callable by anyone other than the `owner`.
  - ☑ Negative test
- Should not allow double mapping of tokens.
  - ☐ Negative test

Function: `_deposit(address _token, address _to, uint256 _amount, byte[] _data, uint256 _gasLimit)`

Facilitate the deposit and transfer of an ERC20 token from layer 1 to layer 2.

Inputs

- `_token`
- **Control**: Full control.
  - **Constraints**: Checked `tokenMapping[_token]` ≠ `address(0)`.
  - **Impact**: The ERC20 token to be deposited.
- `_to`
- **Control**: Full control.
  - **Constraints**: N/A.

  - **Impact**: The destination address for the deposit.
- `_amount`
- **Control**: Full control.
  - **Constraints**: N/A.
  - **Impact**: The amount of the ERC20 token to be deposited.
- `_data`
- **Control**: Full control.
  - **Constraints**: N/A.
  - **Impact**: The data to be passed to the recipient.
- `_gasLimit`
  - **Control**: Full control.
  - **Constraints**: N/A.
  - **Impact**: The gas limit for the deposit.

Branches and code coverage (including function calls)

Intended branches

- Ensure the difference of after and before balance is positive.
  ☑ Test coverage
- Should decrease the balance of `_token` by `_amount` for the `msg.sender`.
  ☑ Test coverage
- Should increase the balance of `_token` by `after - before` for the `address(this)`.
  ☑ Test coverage
- Should forward the payload on to the messenger.
  ☑ Test coverage

Negative behavior

- Should not permit depositing fictitious tokens.
  ☑ Negative test; should be covered by the fact that the token must be registered in the `tokenMapping` mapping.

Function call analysis

- `IERC20Upgradeable(_token).safeTransferFrom(_from, address(this), _amount);`
- **What is controllable?**: `_token`, `_amount`.
  - **If return value controllable, how is it used and how can it go wrong?**: N/A; assumed that the `token` has been previously validated.

  - **What happens if it reverts, reenters, or does other unusual control flow?**: Transfer fails and the function reverts.
- `IL1ScrollMessenger(messenger).sendMessage{value: msg.value}(counterpart, 0, _message, _gasLimit)`
  - **What is controllable?** `_message` (partially), `_gasLimit`.
  - **If return value controllable, how is it used and how can it go wrong?** N/A.
  - **What happens if it reverts, reenters, or does other unusual control flow?** Means not enough fees have been paid to cover the gas cost of the message on L2; function reverts.

5.2 Module: L1ERC721Gateway.sol

Function: `finalizeBatchWithdrawERC721(address _l1Token, address _l2Token, address _from, address _to, uint256[] _tokenIds)`

Allow the batch withdrawal of multiple `tokenIds` of the same ERC721 token from L2 to L1.

Inputs

- `_l1Token`
  - **Control**: Fully controlled by the counterpart.
  - **Constraints**: Checked that `l2Token == tokenMapping[_l1Token]`.
  - **Impact**: The token to be withdrawn.
- `_l2Token`
  - **Control**: Fully controlled by the counterpart.
  - **Constraints**: Checked that `l2Token == tokenMapping[_l1Token]`.
  - **Impact**: The l2Token that was burned on L2.
- `_from`
  - **Control**: Fully controlled by the counterpart.
  - **Constraints**: N/A.
  - **Impact**: The sender of the message on L2.
- `_to`
  - **Control**: Fully controlled by the counterpart.
  - **Constraints**: N/A.
  - **Impact**: The recipient of the token on L1.
- `_tokenIds`
  - **Control**: Fully controlled by the counterpart.
  - **Constraints**: N/A.

- **Impact**: The token IDs to be withdrawn.

Branches and code coverage (including function calls)

Intended branches

- Assumes the same `tokenIds` have been burned on L2.
  ☑ Test coverage
- Assumes the counterpart does no malicious calls.
  ☐ Test coverage
- Increases the balance of the `to` on L1 with the `tokenIds`.
  ☑ Test coverage
- Decreases the balance of the `address(this)` on L1 with the `tokenIds`.
  ☑ Test coverage
- Assumes that `to` is either an `EOA` or that it has inherited the `ERC721Receiver` interface.
  ☑ Test coverage

Negative behavior

- Should not be callable by anyone other than the counterpart.
  ☑ Negative test
- Should not be callable if the `_l2Token` does not match the `tokenMapping[_l1Token]`.
  ☐ Negative test
- Should not allow calling if `l2Token == 0`.
  ☐ Negative test

Function call analysis

- `IERC721Upgradeable(_l1Token).safeTransferFrom(address(this), _to, _tokenIds[i])`
  - **What is controllable?** `_l1Token`, `_to`, `_tokenIds[i]`.
  - **If return value controllable, how is it used and how can it go wrong?** N/A.
  - **What happens if it reverts, reenters, or does other unusual control flow?** Means `_to` cannot accept the token.

Function: `finalizeWithdrawERC721(address _l1Token, address _l2Token, address _from, address _to, uint256 _tokenId)`

Allow the withdrawal of an ERC721 token from L2 to L1.

Inputs

- `_l1Token`
  - **Control**: Fully controlled by the counterpart.
  - **Constraints**: Checked that `l2Token == tokenMapping[_l1Token]`.
  - **Impact**: The token to be withdrawn.
- `_l2Token`
  - **Control**: Fully controlled by the counterpart.
  - **Constraints**: Checked that `l2Token == tokenMapping[_l1Token]`.
  - **Impact**: The l2Token that was burned on L2.
- `_from`
  - **Control**: Fully controlled by the counterpart.
  - **Constraints**: N/A.
  - **Impact**: The sender of the message on L2.
- `_to`
  - **Control**: Fully controlled by the counterpart.
  - **Constraints**: N/A.
  - **Impact**: The recipient of the token on L1.
- `_tokenId`
  - **Control**: Fully controlled by the counterpart.
  - **Constraints**: N/A.
  - **Impact**: The token to be withdrawn.

Branches and code coverage (including function calls)

Intended branches

- Assumes the same `tokenId` has been burned on L2.
  ☑ Test coverage
- Assumes the counterpart does no malicious calls.
  ☑ Test coverage
- Increases the balance of the `to` on L1 with the `tokenId`.
  ☑ Test coverage
- Decreases the balance of the `address(this)` on L1 with the `tokenId`.
  ☑ Test coverage
- Assumes that `to` is either an `EOA` or that it has inherited the `ERC721Receiver` interface.
  ☑ Test coverage

Negative behavior

- Should not be callable by anyone other than the counterpart.
  ☑ Negative test
- Should not be callable if the `_l2Token` does not match the `tokenMapping[_l1Token]`.
  ☐ Negative test
- Should not allow calling if `l2Token == 0`.
  ☐ Negative test

Function call analysis

- `IERC721Upgradeable(_l1Token).safeTransferFrom(address(this), _to, _tokenId)`
  - **What is controllable?** `_l1Token`, `_to`, `_tokenId`.
  - **If return value controllable, how is it used and how can it go wrong?** N/A.
  - **What happens if it reverts, reenters, or does other unusual control flow?** Means `_to` cannot accept the token.

Function: `initialize(address _counterpart, address _messenger)`

Initialize the storage variables as well as the inherited contracts'.

Inputs

- `_counterpart`
  - **Control**: Fully controlled.
  - **Constraints**: N/A.
  - **Impact**: The address of the counterpart.
- `_messenger`
  - **Control**: Fully controlled.
  - **Constraints**: N/A.
  - **Impact**: Scroll messenger address.

Branches and code coverage (including function calls)

Intended branches

- Should call the `OwnableUpgradeable.__Ownable_init()` function.
  ☑ Test coverage
- Should call the `ScrollGatewayBase._initialize()` function.
  ☑ Test coverage
- Should call the `ERC721HolderUpgradeable.__ERC721Holder_init()` function.
  ☐ Test coverage

- Should set the counterpart to `_counterpart`.
  ☑ Test coverage
- Should set the messenger to `_messenger`.
  ☑ Test coverage

Negative behavior

- Should not be callable twice.
  ☑ Negative test

Function: `updateTokenMapping(address _l1Token, address _l2Token)`

Update the mapping of layer 1 to layer 2 token.

Inputs

- `_l1Token`
  - **Control**: Full control.
  - **Constraints**: N/A.
  - **Impact**: The layer 1 token to be approved.
- `_l2Token`
  - **Control**: Full control.
  - **Constraints**: Checked that `_l2Token` $\neq$ `address(0)`.
  - **Impact**: The layer 2 token to be approved.

Branches and code coverage (including function calls)

Intended branches

- Assumed this should not change that often, or at least not during important bridge operations.
  ☐ Test coverage
- Update the state of the `tokenMapping` to reflect the addition.
  ☑ Test coverage

Negative behavior

- Should not be callable by anyone other than the `owner`.
  ☑ Negative test
- Should not allow double mapping of tokens.
  ☐ Negative test

Function: `_batchDepositERC721(address _token, address _to, uint256[] calldata _tokenIds, uint256 _gasLimit)`

Facilitates the batch deposit of ERC721 tokens to L2.

Inputs

- `_token`
  - **Control**: Full control.
  - **Constraints**: N/A.
  - **Impact**: The token to be deposited.
- `_to`
  - **Control**: Full control.
  - **Constraints**: N/A.
  - **Impact**: The recipient of the token on L2.
- `_tokenIds`
  - **Control**: Full control.
  - **Constraints**: N/A.
  - **Impact**: The token IDs to be deposited.
- `_gasLimit`
  - **Control**: Full control.
  - **Constraints**: N/A.
  - **Impact**: The gas limit to be used for the deposit on L2.

Branches and code coverage (including function calls)

Intended branches

- The `tokenIDs` of the token is transferred from the `msg.sender` to this contract.
  ☑ Test coverage
- Assure the `_token` is a valid one and cannot overwrite the mapping in anyway.
  ☑ Test coverage
- The `finalizeBatchDepositERC721` function is called on the L2ERC721Gateway contract.
  ☑ Test coverage
- The `sendMessage` function is called on the L1ScrollMessenger contract.
  ☑ Test coverage
- Assumes there is an expiry associated with the transfer over L2. Currently, this is not the case.
  ☐ Test coverage

Negative behavior

- Should not allow arbitrary tokens to be deposited for legitimate l2 tokens.
  ☑ Negative test
- Should not allow user to deposit more than they have.
  ☑ Negative test

Function call analysis

- `IERC721Upgradeable(_token).safeTransferFrom(msg.sender, address(this), _tokenIds[i])`
- **What is controllable?** `_token`, `_tokenIds[i]`.
  - **If return value controllable, how is it used and how can it go wrong?** N/A.
  - **What happens if it reverts, reenters, or does other unusual control flow?** Means user does not actually own the token; function reverts.
- `IL1ScrollMessenger(messenger).sendMessage{value: msg.value}(counterpart, 0, _message, _gasLimit)`
  - **What is controllable?** `_message` (partly), `_gasLimit`, `msg.value`.
  - **If return value controllable, how is it used and how can it go wrong?** N/A.
  - **What happens if it reverts, reenters, or does other unusual control flow?** If it reverts, it means user has not sent enough `msg.value` to cover for the gas cost of the deposit on L2.

Function: `_depositERC721(address _token, address _to, uint256 _tokenId, uint256 _gasLimit)`

Deposit an ERC721 token to layer 2.

Inputs

- `_token`
  - **Control**: Full control.
  - **Constraints**: N/A.
  - **Impact**: The token to be deposited.
- `_to`
  - **Control**: Full control.
  - **Constraints**: N/A.
  - **Impact**: The recipient of the token on L2.
- `_tokenId`
  - **Control**: Full control.
  - **Constraints**: N/A.
  - **Impact**: The token ID to be deposited.

- `_gasLimit`
  - **Control**: Full control.
  - **Constraints**: N/A.
  - **Impact**: The gas limit to be used for the deposit on L2.

Branches and code coverage (including function calls)

Intended branches

- The `tokenID` of the token is transferred from the `msg.sender` to this contract.
  ☑ Test coverage
- Assure the `_token` is a valid one and cannot overwrite the mapping in anyway.
  ☑ Test coverage
- The `finalizeDepositERC721` function is called on the L2ERC721Gateway contract.
  ☑ Test coverage
- The `sendMessage` function is called on the L1ScrollMessenger contract.
  ☑ Test coverage
- Assumes there is an expiry associated with the transfer over L2. Currently, this is not the case.
  ☐ Test coverage

Negative behaviour

- Should technically not be callable on the same token twice, unless it has been transferred back from L2.
  ☑ Negative test
- Should not allow arbitrary tokens to be deposited for legitimate l2 tokens.
  ☑ Negative test

Function call analysis

- `IERC721Upgradeable(_token).safeTransferFrom(msg.sender, address(this), _tokenId)`
- **What is controllable?**: `_token`, `_tokenId`
  - **If return value controllable, how is it used and how can it go wrong?**: n/a
  - **What happens if it reverts, reenters, or does other unusual control flow?**: means user doesn't actually own the token; function reverts.
- `IL1ScrollMessenger(messenger).sendMessage{value: msg.value}(counterpart, 0, _message, _gasLimit)`
  - **What is controllable?**: `_message`(partially), `_gasLimit`
  - **If return value controllable, how is it used and how can it go wrong?**: n/a
  - **What happens if it reverts, reenters, or does other unusual control flow?**:

means not enough fees have been paid to cover the gas cost of the message on L2; function reverts.

5.3 Module: L1ERC1155Gateway.sol

Function: `finalizeBatchWithdrawERC1155(address _l1Token, address _l2Token, address _from, address _to, uint256[] _tokenIds, uint256[] _amounts)`

Allow the batch withdrawal of multiple `tokenIds` and `amounts` of the same ERC1155 token from L2 to L1.

Inputs

- `_l1Token`
  - **Control**: Fully controlled by the counterpart.
  - **Constraints**: Checked that `l2Token == tokenMapping[_l1Token]`.
  - **Impact**: The token to be withdrawn.
- `_l2Token`
  - **Control**: Fully controlled by the counterpart.
  - **Constraints**: Checked that `l2Token == tokenMapping[_l1Token]`.
  - **Impact**: The l2token that was burned on L2.
- `_from`
  - **Control**: Fully controlled by the counterpart.
  - **Constraints**: N/A.
  - **Impact**: The sender of the message on L2.
- `_to`
  - **Control**: Fully controlled by the counterpart.
  - **Constraints**: N/A.
  - **Impact**: The recipient of the token on L1.
- `_tokenIds`
  - **Control**: Fully controlled by the counterpart.
  - **Constraints**: N/A.
  - **Impact**: The token IDs to be withdrawn.
- `_amounts`
  - **Control**: Fully controlled by the counterpart.
  - **Constraints**: N/A.
  - **Impact**: The amounts of tokens to be withdrawn.

Branches and code coverage (including function calls)

Intended branches

- Assumes the same `tokenIds` amounts have been burned on L2.
  ☑ Test coverage
- Assumes the counterpart does no malicious calls.
  ☑ Test coverage
- Increases the balance of the `to` on L1 with the `tokenIds`.
  ☑ Test coverage
- Decreases the balance of the `address(this)` on L1 with the `tokenIds`.
  ☑ Test coverage
- Assumes that `to` is either an `EOA` or that it has inherited the `ERC1155Receiver` interface.
  ☑ Test coverage
- Check that the lengths of the `_tokenIds` and `_amounts` arrays are the same.
  ☑ Test coverage

Negative behavior

- Should not be callable by anyone other than the counterpart.
  ☑ Negative test
- Should not be callable if the `_l2Token` does not match the `tokenMapping[_l1Token]`.
  ☑ Negative test
- Should not allow calling if `l2Token == 0`.
  ☑ Negative test

Function call analysis

- `IERC1155Upgradeable(_l1Token).safeBatchTransferFrom(address(this), _to, _tokenIds, _amounts, “”)`
  - **What is controllable?** `_l1Token`, `_to`, `_tokenIds`, `_amounts`.
  - **If return value controllable, how is it used and how can it go wrong?** N/A.
  - **What happens if it reverts, reenters, or does other unusual control flow?** It can reenter, due to callbacks, but this is not a problem as the `nonReentrant` modifier is used.

Function: `finalizeWithdrawERC1155(address _l1Token, address _l2Token, address _from, address _to, uint256 _tokenId, uint256 _amount)`

Allow the withdrawal of an `ERC1155` token from L2 to L1.

Inputs

- `_l1Token`
  - **Control**: Fully controlled by the counterpart.
  - **Constraints**: Checked that `l2Token == tokenMapping[_l1Token]`.
  - **Impact**: The token to be withdrawn.
- `_l2Token`
  - **Control**: Fully controlled by the counterpart.
  - **Constraints**: Checked that `l2Token == tokenMapping[_l1Token]`.
  - **Impact**: The l2Token that was burned on L2.
- `_from`
  - **Control**: Fully controlled by the counterpart.
  - **Constraints**: N/A.
  - **Impact**: The sender of the message on L2.
- `_to`
  - **Control**: Fully controlled by the counterpart.
  - **Constraints**: N/A.
  - **Impact**: The recipient of the token on L1.
- `_tokenId`
  - **Control**: Fully controlled by the counterpart.
  - **Constraints**: N/A.
  - **Impact**: The token to be withdrawn.
- `_amount`
  - **Control**: Fully controlled by the counterpart.
  - **Constraints**: N/A.
  - **Impact**: The amount of tokens to be withdrawn.

Branches and code coverage (including function calls)

Intended branches

- Assumes the same `tokenId` amount has been burned on L2.
  ☑ Test coverage
- Assumes the counterpart does no malicious calls.
  ☑ Test coverage
- Increases the balance of the `to` on L1 with the `tokenId` by the `amount`.
  ☑ Test coverage
- Decreases the balance of the `address(this)` on L1 with the `tokenId` by the `amount`.
  ☑ Test coverage
- Assumes that `to` is either an `EOA` or that it has inherited the `ERC1155Receiver` in-

terface.
  ☑ Test coverage

Negative behavior

- Should not be callable by anyone other than the counterpart.
  ☑ Negative test
- Should not be callable if the `_l2Token` does not match the `tokenMapping[_l1Token]`.
  ☑ Negative test
- Should not allow calling if `l2Token == 0`.
  ☑ Negative test

Function call analysis

- `IERC1155Upgradeable(_l1Token).safeTransferFrom(address(this), _to, _tokenId, _amount, "")`
  - **What is controllable?** `_l1Token`, `_to`, `_tokenId`, `_amount`.
  - **If return value controllable, how is it used and how can it go wrong?** N/A.
  - **What happens if it reverts, reenters, or does other unusual control flow?** It can reenter, due to callbacks, but this is not a problem as the `nonReentrant` modifier is used.

Function: `initialize(address _counterpart, address _messenger)`

Initialize the storage variables as well as the inherited contracts'.

Inputs

- `_counterpart`
  - **Control**: Fully controlled.
  - **Constraints**: N/A.
  - **Impact**: The address of the counterpart.
- `_messenger`
  - **Control**: Fully controlled.
  - **Constraints**: N/A.
  - **Impact**: Scroll messenger address.

Branches and code coverage (including function calls)

Intended branches

- Should call the `OwnableUpgradeable.__Ownable_init()` function.

☑ Test coverage

- Should call the `ScrollGatewayBase._initialize()` function.
  ☑ Test coverage
- Should call the `ERC1155HolderUpgradeable.__ERC1155Holder_init()` function.
  ☐ Test coverage
- Should set the counterpart to `_counterpart`.
  ☑ Test coverage
- Should set the messenger to `_messenger`.
  ☑ Test coverage

Negative behavior

- Should not be callable twice.
  ☑ Negative test

Function: `updateTokenMapping(address _l1Token, address _l2Token)`

Update the mapping of layer 1 to layer 2 token.

Inputs

- `_l1Token`
  - **Control**: Full control.
  - **Constraints**: N/A.
  - **Impact**: The layer 1 token to be approved.
- `_l2Token`
  - **Control**: Full control.
  - **Constraints**: Checked that `_l2Token` $\neq$ `address(0)`.
  - **Impact**: The layer 2 token to be approved.

Branches and code coverage (including function calls)

Intended branches

- Assumed this should not change that often, or at least not during important bridge operations.
  ☐ Test coverage
- Update the state of the `tokenMapping` to reflect the addition.
  ☑ Test coverage

Negative behavior

- Should not be callable by anyone other than the `owner`.

☑ Negative test

- Should not allow double mapping of tokens.
  ☐ Negative test

Function: `_batchDepositERC1155(address _token, address _to, uint256[] calldata _tokenIds, uint256[] calldata _amounts, uint256 _gasLimit)`

Facilitates the batch deposit of ERC1155 tokens to L2.

Inputs

- `_token`
  - **Control**: Full control.
  - **Constraints**: N/A.
  - **Impact**: The token to be deposited.
- `_to`
  - **Control**: Full control.
  - **Constraints**: N/A.
  - **Impact**: The recipient of the token on L2.
- `_tokenIds`
  - **Control**: Full control.
  - **Constraints**: Checked the length of `_tokenIds` and `_amounts` are the same.
  - **Impact**: The token IDs to be deposited.
- `_amounts`
  - **Control**: Full control.
  - **Constraints**: Checked amount of tokens to be deposited to be greater than 0 - also that the length of `_tokenIds` and `_amounts` are the same.
  - **Impact**: The amounts of tokens to be deposited.
- `_gasLimit`
  - **Control**: Full control.
  - **Constraints**: N/A.
  - **Impact**: The gas limit to be used for the deposit on L2.

Branches and code coverage (including function calls)

Intended branches

- The `tokenIDs amounts` of the token is transferred from the `msg.sender` to this contract.
  ☑ Test coverage
- Assure the `_token` is a valid one and cannot overwrite the mapping in anyway.

☑ Test coverage

- The `finalizeBatchDepositERC1155` function is called on the L2ERC1155Gateway contract.
  ☑ Test coverage
- The `sendMessage` function is called on the `L1ScrollMessenger` contract.
  ☑ Test coverage
- Assure that the lengths of `_tokenIds` and `_amounts` are the same.
  ☑ Test coverage
- Assumes there is an expiry associated with the transfer over L2. Currently, this is not the case.
  ☐ Test coverage

Negative behavior

- Should not allow arbitrary tokens to be deposited for legitimate l2 tokens.
  ☑ Negative test
- Should not allow user to deposit more than they have.
  ☑ Negative test

Function call analysis

- `IERC1155Upgradeable(_token).safeBatchTransferFrom(msg.sender, address(this), _tokenIds, _amounts, "")`
- **What is controllable?** `_token`, `_tokenIds`, `_amounts`.
  - **If return value controllable, how is it used and how can it go wrong?** N/A.
  - **What happens if it reverts, reenters, or does other unusual control flow?** Should not reenter; if it reverts, it means user cannot afford deposit.
- `IL1ScrollMessenger(messenger).sendMessage{value: msg.value}(counterpart, 0, _message, _gasLimit)`
  - **What is controllable?** `_message` (partly), `_gasLimit`, `msg.value`.
  - **If return value controllable, how is it used and how can it go wrong?** N/A.
  - **What happens if it reverts, reenters, or does other unusual control flow?** If it reverts, it means user has not sent enough `msg.value` to cover for the gas cost of the deposit on L2.

Function: `_depositERC1155(address _token, address _to, uint256 _tokenId, uint256 _amount, uint256 _gasLimit)`

Facilitates the deposit of ERC1155 tokens to L2.

Inputs

- `_token`
  - **Control**: Full control.
  - **Constraints**: N/A.
  - **Impact**: The token to be deposited.
- `_to`
  - **Control**: Full control.
  - **Constraints**: N/A.
  - **Impact**: The recipient of the token on L2.
- `_tokenId`
  - **Control**: Full control.
  - **Constraints**: N/A.
  - **Impact**: The token ID to be deposited.
- `_amount`
  - **Control**: Full control.
  - **Constraints**: Checked amount is not zero.
  - **Impact**: The amount of tokens to be deposited.
- `_gasLimit`
  - **Control**: Full control.
  - **Constraints**: N/A.
  - **Impact**: The gas limit to be used for the deposit on L2.

Branches and code coverage (including function calls)

Intended branches

- The `tokenID` `amount` of the token is transferred from the `msg.sender` to this contract.
  ☑ Test coverage
- Assure the `_token` is a valid one and cannot overwrite the mapping in any way.
  ☑ Test coverage
- The `finalizeDepositERC1155` function is called on the L2ERC1155Gateway contract.
  ☑ Test coverage
- The `sendMessage` function is called on the `L1ScrollMessenger` contract.
  ☑ Test coverage
- Assumes there is an expiry associated with the transfer over L2. Currently, this is not the case.
  ☐ Test coverage

Negative behavior

- Should not allow arbitrary tokens to be deposited for legitimate l2 tokens.
  ☑ Negative test
- Should not allow user to deposit more than they have.
  ☑ Negative test

Function call analysis

- `IERC1155Upgradeable(_token).safeTransferFrom(msg.sender, address(this), _tokenId, _amount, "")`
- **What is controllable?** `_token`, `_tokenId`, `_amount`.
  - **If return value controllable, how is it used and how can it go wrong?** N/A.
  - **What happens if it reverts, reenters, or does other unusual control flow?** Should not reenter; if it reverts, it means user cannot afford deposit.
- `IL1ScrollMessenger(messenger).sendMessage{value: msg.value}(counterpart, 0, _message, _gasLimit)`
  - **What is controllable?** `_message` (partly), `_gasLimit`, `msg.value`.
  - **If return value controllable, how is it used and how can it go wrong?** N/A.
  - **What happens if it reverts, reenters, or does other unusual control flow?** If it reverts, it means user has not sent enough `msg.value` to cover for the gas cost of the deposit on L2.

5.4 Module: L1ETHGateway.sol

Function: `finalizeWithdrawETH(address _from, address _to, uint256 _amount, byte[] _data)`

Finalizes the withdrawal of ETH from L2 to L1. This function is called by the counterpart.

Inputs

- `_from`
  - **Control**: Full controlled.
  - **Constraints**: N/A.
  - **Impact**: Source of ETH from L2.
- `_to`
  - **Control**: Full controlled.
  - **Constraints**: N/A.
  - **Impact**: Destination of ETH.

- `_amount`
  - **Control**: Full controlled.
  - **Constraints**: N/A.
  - **Impact**: Amount of ETH to withdraw.
- `_data`
  - **Control**: Full controlled.
  - **Constraints**: No constraints here. This is critical since it is important that it properly defends against arbitrary calls. Important to use an interface here, so safeguard against arbitrary contract type calls.
  - **Impact**: Calldata to be passed to `_to`.

Branches and code coverage (including function calls)

Intended branches

- Increase the balance of `_to` by `_amount`.
  ☑ Test coverage
- Decrease the balance of `_from` by `_amount`.
  ☑ Test coverage
- Pass the call to `_to` with `_data` as calldata.
  ☑ Test coverage
- Should check that `msg.sender == _amount`. Currently, this is not checked.
  ☑ Test coverage

Negative behavior

- Assumes no malicious intent on the behalf of the counterpart.
  ☐ Negative test

Function call analysis

- `_to.call{value: _amount}("")`
  - **What is controllable?** `_to` and `_amount` - also the calldata `_data`, but this is not used at the moment.
  - **If return value controllable, how is it used and how can it go wrong?** Return value tells whether the call succeeded or not. If it fails, the ETH is not transferred.
  - **What happens if it reverts, reenters, or does other unusual control flow?** If it reverts, the ETH is not transferred. It can reenter and use the `L1ETHGateway` as a proxy to transfer ETH to any address. At the moment, this is not an issue, since calldata is not used, but this should be fixed in the future by using an interface.

Function: `initialize(address _counterpart, address _router, address _messenger)`

Initialize the storage variables as well as the inherited contracts'.

Inputs

- `_counterpart`
  - **Control**: Fully controlled.
  - **Constraints**: N/A.
  - **Impact**: The address of the counterpart.
- `_messenger`
  - **Control**: Fully controlled.
  - **Constraints**: N/A.
  - **Impact**: Scroll messenger address.
- `_router`
  - **Control**: Fully controlled.
  - **Constraints**: N/A.
  - **Impact**: Router address.

Branches and code coverage (including function calls)

Intended branches

- Should call the `ScrollGatewayBase._initialize()` function.
  ☑ Test coverage
- Should set the counterpart to `_counterpart`.
  ☑ Test coverage
- Should set the messenger to `_messenger`.
  ☑ Test coverage
- Should set the router to `_router`.
  ☑ Test coverage

Negative behavior

- Should not be callable twice.
  ☑ Negative test

Function: `_deposit(address _to, uint256 _amount, byte[] _data, uint256 _gasLimit)`

Facilitate the deposit of native tokens from L1 to L2.

Inputs

- `_to`
  - **Control**: Full control.
  - **Constraints**: N/A.
  - **Impact**: The destination address for the deposit.
- `_amount`
  - **Control**: Full control.
  - **Constraints**: N/A.
  - **Impact**: The amount of native tokens to be deposited.
- `_data`
  - **Control**: Full control.
  - **Constraints**: N/A.
  - **Impact**: The data to be passed to the recipient.
- `_gasLimit`
  - **Control**: Full control.
  - **Constraints**: N/A.
  - **Impact**: The gas limit for the deposit.

Branches and code coverage (including function calls)

Intended branches

- Should forward the payload on to the messenger.
  ☑ Test coverage
- Decrease the balance of `msg.sender` by `_amount`.
  ☑ Test coverage
- Increase the balance of `messenger` by `_amount`.
  ☑ Test coverage

Negative behavior

- Should not allow `_amount` to be ≥ `msg.value`. Basically, it should ensure that the user covers the deposits.
  ☑ Negative test

Function call analysis

- `IL1ScrollMessenger(messenger).sendMessage{value: msg.value}(counterpart, _amount, _message, _gasLimit)`
  - **What is controllable?** `_message` (partially), `_gasLimit`.
  - **If return value controllable, how is it used and how can it go wrong?** N/A.

- **What happens if it reverts, reenters, or does other unusual control flow?** Means not enough fees have been paid to cover the gas cost of the message on L2; function reverts.

5.5 Module: L1GatewayRouter.sol

Function: `depositETHAndCall(address _to, uint256 _amount, byte[] _data, uint256 _gasLimit)`

Allows user to deposit ETH and call a contract with the deposited ETH on L2.

Inputs

- `_to`
  - **Control**: Full control.
  - **Constraints**: N/A.
  - **Impact**: The destination address for the deposit.
- `_amount`
  - **Control**: Full control.
  - **Constraints**: Should check that `msg.value` ≥ `_amount`.
  - **Impact**: The amount of the ERC20 native.
- `_data`
  - **Control**: Full control.
  - **Constraints**: N/A.
  - **Impact**: The data to be passed to the recipient.
- `_gasLimit`
  - **Control**: Full control.
  - **Constraints**: N/A.
  - **Impact**: The gas limit for the deposit.

Branches and code coverage (including function calls)

Intended branches

- Allow user to deposit ETH and call a contract with the deposited ETH on L2.
  ☑ Test coverage
- Decrease the balance of ETH for the user.
  ☑ Test coverage
- Increase the balance of ETH for the user on L2.
  ☑ Test coverage

- Increase the balance of the messenger contract.
  ☑ Test coverage

Negative behavior

- Should not allow calling with `msg.value < amount`. This in fact should be checked at the level of `sendMessage` in `L1ETHGateway`.
  ☑ Negative test
- Should not allow calls to arbitrary addresses on the L2 side; this has to be enforced project wide.
  ☑ Negative test

Function call analysis

- `IL1ETHGateway(_gateway).depositETHAndCall{value: msg.value}(_to, _amount, _routerData, _gasLimit);`
  - **What is controllable?** `msg.value`, `_to`, `_amount`.
  - **If return value controllable, how is it used and how can it go wrong?** N/A.
  - **What happens if it reverts, reenters, or does other unusual control flow?** Should refund the user.

Function: `initialize(address _ethGateway, address _defaultERC20Gateway)`

Initialize the storage variables, as well as the inherited contracts'.

Inputs

- `_ethGateway`
- **Control**: Full control.
  - **Constraints**: Checked for nonzero.
  - **Impact**: Sets the `ethGateway` variable.
- `_defaultERC20Gateway`
  - **Control**: Full control.
  - **Constraints**: Checked for nonzero.
  - **Impact**: Sets the `defaultERC20Gateway` variable.

Branches and code coverage (including function calls)

Intended branches

- Should set the default ERC20 gateway.
  ☑ Test coverage

- Should set the ETH gateway.
  ☑ Test coverage
- Call to `OwnableUpgradeable.__Ownable_init()` should succeed.
  ☑ Test coverage

Negative behavior

- Should not be callable twice.
  ☑ Negative test

Function: `setDefaultERC20Gateway(address _defaultERC20Gateway)`

Sets the address of the default ERC20 gateway contract.

Inputs

- `_defaultERC20Gateway`
  - **Control**: Full control.
  - **Constraints**: None.
  - **Impact**: The address of the default ERC20 gateway contract.

Branches and code coverage (including function calls)

Intended branches

- Should update the state of the `defaultERC20Gateway` variable.
  ☑ Test coverage
- Assumes the contract will not be left in an inconsistent state after the change.
  ☐ Test coverage

Negative behavior

- Should not be callable by anyone other than the owner.
  ☑ Negative test

Function: `setERC20Gateway(address[] _tokens, address[] _gateways)`

Updates the mapping from token address to gateway address.

Inputs

- `_tokens`
  - **Control**: Full control.
  - **Constraints**: Length of `_tokens` must match length of `_gateways`.

  - **Impact**: The list of addresses of tokens to update.
- `_gateways`
  - **Control**: Full control.
  - **Constraints**: Length of `_gateways` must match length of `_tokens`.
  - **Impact**: The list of addresses of gateways that will be accesible.

Branches and code coverage (including function calls)

Intended branches

- Should update the state of the `ERC20Gateway` mapping for each token in `_tokens` with the corresponding gateway in `_gateways`.
  ☑ Test coverage
- Assumes the contract will not be left in an inconsistent state after the change.
  ☐ Test coverage

Negative behavior

- Should only be callable by the owner.
  ☑ Negative test

Function: `setETHGateway(address _ethGateway)`

Sets the address of the ETH gateway contract.

Inputs

- `_ethGateway`
  - **Control**: Full control.
  - **Constraints**: None.
  - **Impact**: The address of the ETH gateway contract.

Branches and code coverage (including function calls)

Intended branches

- Should update the state of the `ethGateway` variable.
  ☑ Test coverage
- Assume the contract will not be left in an inconsistent state after the change.
  ☐ Test coverage

Negative behavior

- Should not be callable by anyone other than the owner.

☑ Negative test

5.6 Module: L1MessageQueue.sol

Function: `appendCrossDomainMessage(address _target, uint256 _gasLimit, byte[] _data)`

Appends a cross-domain message to the queue.

Inputs

- `_target`
  - **Control**: Only messenger.
  - **Constraints**: None.
  - **Impact**: The destination on L2.
- `_gasLimit`
  - **Control**: Only messenger.
  - **Constraints**: None.
  - **Impact**: The gas limit for the message.
- `_data`
  - **Control**: Only messenger.
  - **Constraints**: None.
  - **Impact**: The data to be passed to the L2 destination.

Branches and code coverage (including function calls)

Intended branches

- Append a valid message to the queue.
  ☐ Test coverage
- Assumes the message is legitimate and previous checks have been performed.
  ☐ Test coverage
- Emit `QueueTransaction` event.
  ☐ Test coverage

Negative behavior

- Should not be callable by anyone other than the messenger.
  ☑ Negative test
- Should NOT be callable as arbitrary call by the messenger. This needs enforcement at the messenger level.
  ☐ Negative test

Function: `initialize(address _messenger, address _gasOracle)`

Initializes the contract.

Inputs

- `_messenger`
  - **Control**: Full.
  - **Constraints**: None.
  - **Impact**: The messenger contract address.
- `_gasOracle`
  - **Control**: Full.
  - **Constraints**: None.
  - **Impact**: The gas price oracle contract address.

Branches and code coverage (including function calls)

Intended branches

- Set the messenger address.
  ☑ Test coverage
- Set the gas oracle address.
  ☑ Test coverage

Negative behavior

- Call all underlying initializers.
  ☑ Negative test
- Should not be callable multiple times.
  ☑ Negative test

Function: `updateGasOracle(address _newGasOracle)`

Updates the address of the gas oracle.

Inputs

- `_newGasOracle`
  - **Control**: Only owner.
  - **Constraints**: None.
  - **Impact**: The new gas oracle address.

Branches and code coverage (including function calls)

Intended branches

- Should update the `gasOracle` variable.
  ☑ Test coverage
- Should emit `UpdateGasOracle` event.
  ☑ Test coverage
- Assure that `gasOracle` is different than `_newGasOracle`.
  ☐ Test coverage
- Assure no implications of the gas oracle address being changed.
  ☑ Test coverage

Negative behavior

- Should not be callable by anyone other than the owner.
  ☑ Negative test

5.7 Module: L1ScrollMessenger.sol

Function: `initialize(address _counterpart, address _feeVault, address _rollup, address _messageQueue)`

Initializes the storage of L1ScrollMessenger.

Inputs

- `_counterpart`
  - **Control**: Full control.
  - **Constraints**: None.
  - **Impact**: The address of counterpart.
- `_feeVault`
  - **Control**: Full control.
  - **Constraints**: None.
  - **Impact**: The address of the vault where fees are sent.
- `_rollup`
  - **Control**: Full control.
  - **Constraints**: None.
  - **Impact**: The address of ScrollChain contract.
- `_messageQueue`
  - **Control**: Full control.
  - **Constraints**: None.

- **Impact**: Address of message queue.

Branches and code coverage (including function calls)

Intended branches

- Set all important variables.
  ☑ Test coverage
- Call all underlying initializer functions.
  ☑ Test coverage

Negative behavior

- Should not be callable multiple times.
  ☑ Negative test

Function: `relayMessageWithProof(address _from, address _to, uint256 _value, uint256 _nonce, byte[] _message, L2MessageProof _proof)`

Should relay a message from L2 to L1, by verifying the proof and adding the message to the queue.

Inputs

- `_from`
  - **Control**: Partial control
  - **Constraints**: Checked that it's not the same as `xDomainMsgSender`
  - **Impact**: The address the message has been sent from on L2.
- `_to`
  - **Control**: Partial control.
  - **Constraints**: Limited; only checked that it is not `address(this)` or `messageQueue`. Should be checked against an interface.
  - **Impact**: The address where the call will be executed on L1.
- `_value`
  - **Control**: Partial control.
  - **Constraints**: N/A.
  - **Impact**: The value to be sent with the call.
- `_nonce`
  - **Control**: Partial control.
  - **Constraints**: N/A.
  - **Impact**: Nonce of the message; should be unique.
- `_message`

  - **Control**: Partial control.
  - **Constraints**: N/A.
  - **Impact**: The calldata to be executed on L1.
- `_proof`
  - **Control**: Partial control.
  - **Constraints**: Not checked currently.
  - **Impact**: Very important. The proof should be valid and should match the message.

Branches and code coverage (including function calls)

Intended branches

- Should ensure that the proofs are valid and that they match the message.
  ☑ Test coverage
- Ensure that message is not a duplicate nor that it has been sent before.
  ☑ Test coverage
- Update the `isL2MessageExecuted` state after the message has been executed successfully.
  ☑ Test coverage
- Should ensure that the message is not too old. Currently not implemented.
  ☐ Test coverage
- Ensure that the execution of the call is successful.
  ☐ Test coverage

Negative behavior

- Should really not allow calls on arbitrary addresses. Maybe enforcing an interface would be better.
  ☐ Negative test
- Not allow `XDOMAIN_MESSAGE_SENDER` reentrancy.
  ☑ Negative test
- Should not allow execution of messages that do not match the proof. Also, assumes that the proof is legitimate.
  ☑ Negative test

Function call analysis

- `IScrollChain(_rollup).isBatchFinalized(_proof.batchHash)`
- **What is controllable?** `_proof.batchHash`.
  - **If return value controllable, how is it used and how can it go wrong?** Checks if the batch is finalized.

  - **What happens if it reverts, reenters, or does other unusual control flow?** The message may not be added to the queue; should revert.
- `_to.call{value: _value}(_message);`
  - **What is controllable?** `_to`, `_value`, `_message`.
  - **If return value controllable, how is it used and how can it go wrong?** Executes the call.
  - **What happens if it reverts, reenters, or does other unusual control flow?** Then execution will revert, and the message will not be executed. Should revert.

Function: `sendMessage(address _to, uint256 _value, byte[] _message, uint256 _gasLimit, address _refundAddress)`

Allow sending messages from L1 to L2.

Inputs

- `_to`
  - **Control**: Full control.
  - **Constraints**: None.
  - **Impact**: The destination address of the message on L2.
- `_value`
  - **Control**: Full control.
  - **Constraints**: Checked against `msg.value` + estimated fee.
  - **Impact**: The value to be sent to the destination address on L2.
- `_message`
  - **Control**: Full control.
  - **Constraints**: None.
  - **Impact**: The payload of the message.
- `_gasLimit`
  - **Control**: Full control; used in calculation of the fee.
  - **Constraints**: None.
  - **Impact**: The gas limit for the message on L2.
- `_refundAddress`
  - **Control**: Full control.
  - **Constraints**: N/A.
  - **Impact**: The address to refund any excess ETH to.

Branches and code coverage (including function calls)

Intended branches

- Assumes off-chain infrastructure is working fine even if someone was to abuse this function, calling it multiple times with different parameters.
  - ☐ Test coverage
- Should deplete the caller's balance by the amount that is sent to the destination address on L2.
  - ☐ Test coverage
- Assumes the message will be delivered to the destination address on L2.
  - ☐ Test coverage
- Should refund any excess ETH to the `_refundAddress`.
  - ☑ Test coverage
- Should add the crafted message to the queue.
  - ☑ Test coverage
- Should emit the `MessageSent` event.
  - ☑ Test coverage
- Ensure that `msg.value ≥ _value + fee`.
  - ☑ Test coverage

Negative behavior

- Should not be callable when paused.
  - ☐ Negative test
- Should not be able to craft same message twice. Here the `messageNonce` is used as a unique identifier for the message as well as the `isL1MessageSent` mapping.
  - ☑ Negative test

Function call analysis

- `IL1MessageQueue(_messageQueue).nextCrossDomainMessageIndex()`
- **What is controllable?** N/A.
  - **If return value controllable, how is it used and how can it go wrong?** Computes the nonce according to the message queue.
  - **What happens if it reverts, reenters, or does other unusual control flow?** The nonce may be incorrect; message may still be delivered.
- `IL1MessageQueue(_messageQueue).estimateCrossDomainMessageFee(address(this), _counterpart, _xDomainCalldata, _gasLimit)`
- **What is controllable?** `_gasLimit`.
  - **If return value controllable, how is it used and how can it go wrong?** Computes the fee according to the message size and gas limit.

  - **What happens if it reverts, reenters, or does other unusual control flow?** The fee may be incorrect; message may still be delivered.
- `feeVault.call{value: _fee}("")`
- **What is controllable?** N/A.
  - **If return value controllable, how is it used and how can it go wrong?** Sends the fee to the fee vault.
  - **What happens if it reverts, reenters, or does other unusual control flow?** The fee may not be sent to the fee vault; should return false and revert at next line (require).
- `IL1MessageQueue(_messageQueue).appendCrossDomainMessage(_counterpart, _gasLimit, _xDomainCalldata)`
  - **What is controllable?** `gasLimit`.
  - **If return value controllable, how is it used and how can it go wrong?** Adds the message to the queue.
  - **What happens if it reverts, reenters, or does other unusual control flow?** The message may not be added to the queue; should revert.

Function: `setPause(bool _status)`

Pause or unpause the contract.

Inputs

- `_status`
  - **Control**: Full control.
  - **Constraints**: N/A.
  - **Impact**: Dictates the pause status to update.

Branches and code coverage (including function calls)

Intended branches

- Should update the `paused` status.
  ☑ Test coverage

Negative behavior

- Should not be called by anyone other than the owner.
  ☑ Negative test

5.8 Module: L1StandardERC20Gateway.sol

Function: `finalizeWithdrawERC20(address _l1Token, address _l2Token, address _from, address _to, uint256 _amount, byte[] _data)`

Perform the finalization of withdraw ERC20 token from layer 2 to layer 1.

Inputs

- `_l1Token`
  - **Control**: Full control.
  - **Constraints**: Not checked; should check the `tokenMapping` just like in `L1CustomERC20Gateway`.
  - **Impact**: l1Token to be withdrawn.
- `_l2Token`
  - **Control**: Full control.
  - **Constraints**: Not checked; should check the `tokenMapping` just like in `L1CustomERC20Gateway`.
  - **Impact**: l2token that was burned.
- `_from`
  - **Control**: Full control.
  - **Constraints**: N/A (checked in l2).
  - **Impact**: Sender from l2.
- `_to`
  - **Control**: Full control.
  - **Constraints**: N/A.
  - **Impact**: Destination address.
- `_amount`
  - **Control**: Full control.
  - **Constraints**: N/A (checked in l2).
  - **Impact**: Amount to be withdrawn.
- `_data`
  - **Control**: Full control.
  - **Constraints**: N/A (checked in l2).
  - **Impact**: Data to be passed on to the recipient.

Branches and code coverage (including function calls)

Intended branches

- Should ensure that `_l2Token == tokenMapping[_l1Token] && l2Token ≠ address(0)`.
  - ☐ Test coverage

- Increase the balance of `_l1Token` by `_amount` for the `_to`.
  ☑ Test coverage
- Decrease the balance of `_l1Token` by `_amount` for the `address(this)`.
  ☑ Test coverage
- Assume the `counterpart` is legitimate and only calls this function when an identical `amount` of `_l2Token` is burned on layer 2.
  ☑ Test coverage
- Forward the `_data` to the `_to` on layer 2.
  ☑ Test coverage

Negative behavior

- Should not be callable by anyone other than the `counterpart`.
  ☑ Negative test
- Should not be callable multiple times for the same burn.
  ☑ Negative test
- Should not be callable with a zero `_amount`.
  ☑ Negative test

Function call analysis

- `IERC20Upgradeable(_l1Token).safeTransfer(_to, _amount)`
  - **What is controllable?** `_l1Token`.
  - **If return value controllable, how is it used and how can it go wrong?** N/A; assumed that the `l1token` has been previously validated
  - **What happens if it reverts, reenters, or does other unusual control flow?** Transfer fails and the function reverts.

Function: `initialize(address _counterpart, address _router, address _messenger, address _l2TokenImplementation, address _l2TokenFactory)`

Initialize the storage of L1StandardERC20Gateway.

Inputs

- `_counterpart`
  - **Control**: Full control.
  - **Constraints**: N/A.
  - **Impact**: The address of the counterpart.
- `_router`
  - **Control**: Full control.
  - **Constraints**: N/A.

  - **Impact**: The address of the router.
- `_messenger`
  - **Control**: Full control.
  - **Constraints**: N/A.
  - **Impact**: The address of the messenger.
- `_l2TokenImplementation`
  - **Control**: Full control.
  - **Constraints**: N/A.
  - **Impact**: The address of the `l2TokenImplementation`.
- `_l2TokenFactory`
  - **Control**: Full control.
  - **Constraints**: N/A.
  - **Impact**: The address of the `l2TokenFactory`.

Branches and code coverage (including function calls)

Intended branches

- Set the counterpart address.
  ☑ Test coverage
- Set the router address.
  ☑ Test coverage
- Set the messenger address.
  ☑ Test coverage
- Set the `l2TokenImplementation` address.
  ☑ Test coverage
- Set the `l2TokenFactory` address.
  ☑ Test coverage
- Call the initialize function of the base contract.
  ☑ Test coverage

Negative behavior

- Should not be callable twice.
  ☑ Negative test

Function: `_deposit(address _token, address _to, uint256 _amount, byte[] _data, uint256 _gasLimit)`

Facilitate the deposit and transfer of an ERC20 token from layer 1 to layer 2.

Inputs

- `_token`
- **Control**: Full control.
  - **Constraints**: Checked `tokenMapping[_token]` existence.
  - **Impact**: The ERC20 token to be deposited.
- `_to`
- **Control**: Full control.
  - **Constraints**: N/A.
  - **Impact**: The destination address for the deposit.
- `_amount`
- **Control**: Full control.
  - **Constraints**: N/A.
  - **Impact**: The amount of the ERC20 token to be deposited.
- `_data`
- **Control**: Full control.
  - **Constraints**: N/A.
  - **Impact**: The data to be passed to the recipient.
- `_gasLimit`
  - **Control**: Full control.
  - **Constraints**: N/A.
  - **Impact**: The gas limit for the deposit.

Branches and code coverage (including function calls)

Intended branches

- Ensure the difference of the after and before balance is positive.
  ☑ Test coverage
- Should decrease the balance of `_token` by `_amount` for the `msg.sender`.
  ☑ Test coverage
- Should increase the balance of `_token` by `after - before` for the `address(this)`.
  ☑ Test coverage
- Should forward the payload on to the messenger.
  ☑ Test coverage
- Should set the `tokenMapping[_token]` to the `l2Token` if it does not exist yet.
  ☑ Test coverage
- Should accept fee-on-transfer tokens.
  ☑ Test coverage

Negative behavior

- Should not permit depositing fictitious tokens.
  ☑ Negative test
- Should not create `l2Token` if it does not exist yet; it should only compute its address if it is not registered in the `tokenMapping`.
  ☑ Negative test
- Should not permit depositing zero amount.
  ☑ Negative test
- Should ensure that user deposits the exact amount of token.
  ☑ Negative test

Function call analysis

- `IERC20(_token).safeTransferFrom(_from, address(this), _amount);`
- **What is controllable?** `_token`, `_amount`.
  - **If return value controllable, how is it used and how can it go wrong?** N/A.
  - **What happens if it reverts, reenters, or does other unusual control flow?** Transfer fails and the function reverts.
- `_l2Token = getL2ERC20Address(_token)`
- **What is controllable?** `_token`.
  - **If return value controllable, how is it used and how can it go wrong?** Should return the computed predictable address of the L2 token.
  - **What happens if it reverts, reenters, or does other unusual control flow?** Function reverts.
- `IL1ScrollMessenger(messenger).sendMessage{value: msg.value}(counterpart, 0, _message, _gasLimit)`
  - **What is controllable?** `_message` (partially), `_gasLimit`.
  - **If return value controllable, how is it used and how can it go wrong?** N/A.
  - **What happens if it reverts, reenters, or does other unusual control flow?** Means not enough fees have been paid to cover the gas cost of the message on L2; function reverts.

5.9 Module: L1WETHGateway.sol

Function: `finalizeWithdrawERC20(address _l1Token, address _l2Token, address _from, address _to, uint256 _amount, byte[] _data)`

Allows the finalization of a withdraw from L2 to L1.

Inputs

- `_l1Token`
  - **Control**: Full control.
  - **Constraints**: Checked that it is the WETH address.
  - **Impact**: l1Token to be withdrawn.
- `_l2Token`
  - **Control**: Full control
  - **Constraints**: Checked `l2Token == l2WETH`
  - **Impact**: l2Token that was burned.
- `_from`
  - **Control**: Full control.
  - **Constraints**: N/A (checked in l2).
  - **Impact**: Sender from l2.
- `_to`
  - **Control**: Full control.
  - **Constraints**: N/A.
  - **Impact**: Destination address.
- `_amount`
  - **Control**: Full control.
  - **Constraints**: Checked that it is `msg.value`.
  - **Impact**: Amount to be withdrawn.
- `_data`
  - **Control**: Full control.
  - **Constraints**: N/A (checked in l2).
  - **Impact**: Data to be passed on to the recipient.

Branches and code coverage (including function calls)

Intended branches

- `msg.value` should be equal to `_amount`.
  ☑ Test coverage
- `_l1Token` should be equal to `WETH`.
  ☑ Test coverage
- `_l2Token` should be equal to `l2WETH`.
  ☑ Test coverage
- `_amount` should be greater than zero.
  ☑ Test coverage
- Should forward `_data` to `_to`. Maybe there could be an interface for this, such

that it is safer during the call. See finding 3.3
  ☐ Test coverage

Negative behavior

- Should not be callable by anyone other than the `counterpart`
  ☑ Negative test
- Should not be callable multiple times for the same burn action on L2
  ☑ Negative test
- Should not be callable with a zero `_amount`
  ☑ Negative test

Function call analysis

- `IWETH(_l1Token).deposit{value: _amount}();`
- **What is controllable?** `_l1Token`, `_amount`.
  - **If return value controllable, how is it used and how can it go wrong?** N/A.
  - **What happens if it reverts, reenters, or does other unusual control flow?** Should revert.
- `IERC20(_l1Token).safeTransfer(_to, _amount);`
  - **What is controllable?** `_l1Token`, `_to`, `_amount`.
  - **If return value controllable, how is it used and how can it go wrong?** N/A.
  - **What happens if it reverts, reenters, or does other unusual control flow?** Should revert; cannot reenter unless it is ERC777, which should not happen.

Function: `initialize(address _counterpart, address _router, address _messenger)`

Initialize the storage variables as well as the inherited contracts'.

Inputs

- `_counterpart`
  - **Control**: Fully controlled.
  - **Constraints**: N/A.
  - **Impact**: The address of the counterpart.
- `_messenger`
  - **Control**: Fully controlled.
  - **Constraints**: N/A.
  - **Impact**: Scroll messenger address.
- `_router`

- **Control**: Fully controlled.
- **Constraints**: N/A.
- **Impact**: Router address.

Branches and code coverage (including function calls)

Intended branches

- Should call the `ScrollGatewayBase._initialize()` function.
  ☑ Test coverage
- Should set the counterpart to `_counterpart`.
  ☑ Test coverage
- Should set the messenger to `_messenger`.
  ☑ Test coverage
- Should set the router to `_router`.
  ☑ Test coverage

Negative behavior

- Should not be callable twice.
  ☑ Negative test

Function: `receive()`

Receive native Ether from WETH contract.

Branches and code coverage (including function calls)

Intended branches

- Should technically only be called by WETH during unwrap.
  ☑ Test coverage

Negative behavior

- Should not assume that the contract cannot receive Ether in another way (e.g., selfdestruct).
  ☑ Negative test

Function: `_deposit(address _token, address _to, uint256 _amount, byte[] _data, uint256 _gasLimit)`

Facilitate the deposit and transfer of WETH from layer 1 to layer 2.

Inputs

- `_token`
- **Control**: Full control.
  - **Constraints**: Checked it is the WETH address.
  - **Impact**: The ERC20 token to be deposited.
- `_to`
- **Control**: Full control.
  - **Constraints**: N/A.
  - **Impact**: The destination address for the deposit.
- `_amount`
- **Control**: Full control.
  - **Constraints**: N/A.
  - **Impact**: The amount of the ERC20 token to be deposited.
- `_data`
- **Control**: Full control.
  - **Constraints**: N/A.
  - **Impact**: The data to be passed to the recipient.
- `_gasLimit`
  - **Control**: Full control.
  - **Constraints**: N/A.
  - **Impact**: The gas limit for the deposit.

Branches and code coverage (including function calls)

Intended branches

- Should decrease the balance of `WETH` by `_amount` for the `msg.sender`.
  ☑ Test coverage
- Should increase the balance of `native` by `_amount` for the `messenger`. This is because `msg.value` will be used as the `fee` for the `messenger`, whereas the `_amount` will be used as the actual amount to be deposited.
  ☑ Test coverage
- Should forward the payload on to the messenger.
  ☑ Test coverage

Negative behavior

- Should not permit depositing fictitious tokens.
  ☑ Negative test

5.10 Module: L2CustomERC20Gateway.sol

Function: `finalizeDepositERC20(address _l1Token, address _l2Token, address _from, address _to, uint256 _amount, byte[] _data)`

Deposit an ERC token from L1 to L2.

Inputs

- `_l1Token`
  - **Control**: Full control.
  - **Constraints**: Checked `tokenMapping[_l1Token] == _l2Token`.
  - **Impact**: l1Token to be deposited.
- `_l2Token`
  - **Control**: Full control.
  - **Constraints**: Checked `tokenMapping[_l1Token] == _l2Token`.
  - **Impact**: l2Token that was burned.
- `_from`
  - **Control**: Full control.
  - **Constraints**: N/A.
  - **Impact**: Sender from l2.
- `_to`
  - **Control**: Full control.
  - **Constraints**: N/A.
  - **Impact**: Destination address.
- `_amount`
  - **Control**: Full control.
  - **Constraints**: N/A.
  - **Impact**: Amount to be deposited.
- `_data`
  - **Control**: Full control.
  - **Constraints**: N/A.
  - **Impact**: Data to be passed on to the recipient.

Branches and code coverage (including function calls)

Intended branches

- Should ensure that `_l2Token == tokenMapping[_l1Token] && l2Token ≠ address(0)`.
  - ☐ Test coverage

- Mint amount of IScrollERC20Token to the recipient
  ☑ Test coverage
- Decrease the balance of the user on the L1 Side
  ☐ Test coverage

Negative behaviour

- Should not be callable by anyone other than the `counterpart`
  ☑ Negative test

Function call analysis

- `IScrollStandardERC20(_l2Token).mint(_to, _amount)`
  - **What is controllable?** `_to`.
  - **If return value controllable, how is it used and how can it go wrong?** N/A.
  - **What happens if it reverts, reenters, or does other unusual control flow?** N/A.

Function: `updateTokenMapping(address _l1Token, address _l2Token)`

Update the mapping of layer 1 to layer 2 token.

Inputs

- `_l1Token`
  - **Control**: Full control.
  - **Constraints**: N/A.
  - **Impact**: The layer 1 token to be approved.
- `_l2Token`
  - **Control**: Full control.
  - **Constraints**: Checked that `_l2Token` $\neq$ `address(0)`.
  - **Impact**: The layer 2 token to be approved.

Branches and code coverage (including function calls)

Intended branches

- Assumed this should not change that often, or at least not during important bridge operations.
  ☐ Test coverage
- Update the state of the `tokenMapping` to reflect the addition.
  ☑ Test coverage

Negative behavior

- Should not be callable by anyone other than the `owner`.
  ☑ Negative test
- Should not allow double mapping of tokens
  ☐ Negative test

Function: `_withdraw(address _token, address _to, uint256 _amount, byte[] _data, uint256 _gasLimit)`

Facilitate the withdrawal of a L2 token to redeem L1 Tokens.

Inputs

- `_token`
- **Control**: Full control.
  - **Constraints**: Checked `tokenMapping[_token] ≠ address(0)`.
  - **Impact**: The ERC20 token to be withdrawn on the L1 side.
- `_to`
- **Control**: Full control.
  - **Constraints**: N/A.
  - **Impact**: The destination address for the withdrawal.
- `_amount`
- **Control**: Full control.
  - **Constraints**: N/A.
  - **Impact**: The amount of the ERC20 token to be withdrawn.
- `_data`
- **Control**: Full control.
  - **Constraints**: N/A.
  - **Impact**: The data to be passed to the recipient.
- `_gasLimit`
  - **Control**: Full control.
  - **Constraints**: N/A.
  - **Impact**: The gas limit for the withdrawal.

Branches and code coverage (including function calls)

Intended branches

- Ensure the differenece of the after and before balance is positive.

☐ Test coverage

- Should burn `_token` by `_amount` for the `msg.sender`.
  ☐ Test coverage
- Should increase the balance of `_token` of the recipient by amount
  ☑ Test coverage
- Should forward the payload on to the messenger.
  ☐ Test coverage

Negative behavior

- Should not permit withdrawing fictitious tokens.
  ☐ Negative test

Function call analysis

- `IERC20Upgradeable(_token).safeTransferFrom(_from, address(this), _amount);`
- **What is controllable?** `_token`, `_amount`.
  - **If return value controllable, how is it used and how can it go wrong?** N/A; assumed that the `token` has been previously validated.
  - **What happens if it reverts, reenters, or does other unusual control flow?** Transfer fails and the function reverts.
- `IL2ScrollMessenger(messenger).sendMessage{value: msg.value}(counterpart, 0, _message, _gasLimit)`
  - **What is controllable?** `_gasLimit`.
  - **If return value controllable, how is it used and how can it go wrong?** N/A.
  - **What happens if it reverts, reenters, or does other unusual control flow?** N/A (fee finding).

5.11 Module: L2ERC721Gateway.sol

Function: `finalizeBatchDepositERC721(address _l1Token, address _l2Token, address _from, address _to, uint256[] _tokenIds)`

Allow the batch deposit of multiple `tokenIds` of the same ERC721 token from L1 to L2.

Inputs

- `_l1Token`
  - **Control**: Fully controlled by the counterpart.
  - **Constraints**: Checked that `l2Token == tokenMapping[_l1Token]` on L1.

  - **Impact**: The token to be deposited.
- `_l2Token`
  - **Control**: Fully controlled by the counterpart.
  - **Constraints**: Checked that `l2Token == tokenMapping[_l1Token]` on L1.
  - **Impact**: The l2token that was burned on L2.
- `_from`
  - **Control**: Fully controlled by the counterpart.
  - **Constraints**: N/A.
  - **Impact**: The sender of the message on L1.
- `_to`
  - **Control**: Fully controlled by the counterpart.
  - **Constraints**: N/A.
  - **Impact**: The recipient of the token on L2.
- `_tokenIds`
  - **Control**: Fully controlled by the counterpart.
  - **Constraints**: N/A.
  - **Impact**: The token IDs to be deposited.

Branches and code coverage (including function calls)

Intended branches

- Assumes the same `tokenIds` have been transferred on L1.
  ☐ Test coverage
- Increases the balance of the `to` on L2 with the `tokenIds` by minting
  ☑ Test coverage
- Assumes that `to` is either an `EOA` or that it has inherited the `ERC721Receiver` interface.
  ☐ Test coverage

Negative behavior

- Should not be callable by anyone other than the counterpart.
  ☑ Negative test
- Should not be callable if the `_l2Token` does not match the `tokenMapping[_l1Token]` on L1
  ☐ Negative test
- Should not allow calling if `l2Token == 0` on L1
  ☑ Negative test

Function call analysis

- `IScrollERC721(_l2Token).mint(_to, _tokenIds[i])`
  - **What is controllable?** `_to` and `_tokenIds` if checked on L1.
  - **If return value controllable, how is it used and how can it go wrong?** Discarded.
  - **What happens if it reverts, reenters, or does other unusual control flow?** Means `_to` cannot accept the tokens.

Function: `finalizeDepositERC721(address _l1Token, address _l2Token, address _from, address _to, uint256 _tokenId)`

Allow the withdrawal of an ERC721 token from L2 to L1.

Inputs

- `_l1Token`
  - **Control**: Fully controlled by the counterpart.
  - **Constraints**: Checked that `l2Token == tokenMapping[_l1Token]` on L1, not on L2.
  - **Impact**: The L1 token stored in the gateway.
- `_l2Token`
  - **Control**: Fully controlled by the counterpart.
  - **Constraints**: Checked that `l2Token == tokenMapping[_l1Token]` on L1, not on L2.
  - **Impact**: The l2Token that was minted on the L2.
- `_from`
  - **Control**: Fully controlled by the counterpart.
  - **Constraints**: N/A.
  - **Impact**: The sender of the message on L1.
- `_to`
  - **Control**: Fully controlled by the counterpart.
  - **Constraints**: N/A.
  - **Impact**: The recipient of the token on L2.
- `_tokenId`
  - **Control**: Fully controlled by the counterpart.
  - **Constraints**: N/A.
  - **Impact**: The `tokenId` to be deposited.

Branches and code coverage (including function calls)

Intended branches

- Increases the balance of the `to` on L2 with the `tokenId`
  - ☐ Test coverage
- Decreases/store the balance of the `address(this)` on L1 with the `tokenId`
  - ☑ Test coverage

Negative behaviour

- Should not be callable by anyone other than the counterpart.
  - ☑ Negative test
- Should not be callable if the `_l2Token` does not match the `tokenMapping[_l1Token]` on L1
  - ☐ Negative test
- Should not allow calling if `l2Token == 0`.
  - ☑ Negative test

Function call analysis

- `IScrollERC721(_l2Token).mint(_to, _tokenId)`
  - **What is controllable?**: to
  - **If return value controllable, how is it used and how can it go wrong?**: n/a
  - **What happens if it reverts, reenters, or does other unusual control flow?**: Nothing

Function: `initialize(address _counterpart, address _messenger)`

Initialize the storage variables, as well as the inherited contracts'.

Inputs

- `_counterpart`
  - **Control**: Fully controlled.
  - **Constraints**: N/A.
  - **Impact**: The address of the counterpart.
- `_messenger`
  - **Control**: Fully controlled.
  - **Constraints**: N/A.
  - **Impact**: Scroll messenger address.

Branches and code coverage (including function calls)

Intended branches

- Should call the `OwnableUpgradeable.__Ownable_init()` function.
  ☑ Test coverage
- Should call the `ScrollGatewayBase._initialize()` function.
  ☑ Test coverage
- Should call the `ERC721HolderUpgradeable.__ERC721Holder_init()` function.
  ☐ Test coverage
- Should set the counterpart to `_counterpart`.
  ☑ Test coverage
- Should set the messenger to `_messenger`.
  ☑ Test coverage

Negative behavior

- Should not be callable twice.
  ☑ Negative test

Function: `updateTokenMapping(address _l1Token, address _l2Token)`

Update the mapping of layer 1 to layer 2 token.

Inputs

- `_l1Token`
  - **Control**: Full control.
  - **Constraints**: N/A.
  - **Impact**: The layer 1 token to be approved.
- `_l2Token`
  - **Control**: Full control.
  - **Constraints**: Checked that `_l2Token` $\neq$ `address(0)`.
  - **Impact**: The layer 2 token to be approved.

Branches and code coverage (including function calls)

Intended branches

- Assumed this should not change that often, or at least not during important bridge operations.
  ☐ Test coverage
- Update the state of the `tokenMapping` to reflect the addition.

☑ Test coverage

Negative behavior

- Should not be callable by anyone other than the `owner`.
  ☑ Negative test
- Should not allow double mapping of tokens
  ☐ Negative test

Function: `_batchWithdrawERC721(address _token, address _to, uint256[] calldata _tokenIds, uint256 _gasLimit)`

Facilitates the batch withdrawal of ERC721 tokens to L2.

Inputs

- `_token`
  - **Control**: Full control.
  - **Constraints**: N/A.
  - **Impact**: The token to be withdrawn.
- `_to`
  - **Control**: Full control.
  - **Constraints**: N/A.
  - **Impact**: The recipient of the token on L2.
- `_tokenIds`
  - **Control**: Full control.
  - **Constraints**: N/A.
  - **Impact**: The token IDs to be withdrawn.
- `_gasLimit`
  - **Control**: Full control.
  - **Constraints**: N/A.
  - **Impact**: The gas limit to be used for the withdraw on L2.

Branches and code coverage (including function calls)

Intended branches

- The owner of the tokenIds is valid.
  ☑ Test coverage
- Token is in the l1Token mapping list
  ☑ Test coverage
- The `finalizeBatchWithdrawERC721` function is called on the `L2ERC721Gateway` con-

tract.
  - ☐ Test coverage
- The `sendMessage` function is called on the L2ScrollMessenger contract.
  - ☐ Test coverage

Negative behaviour

- Should not allow arbitrary tokens to be withdrawed for legitimate l1 ERC721s
  - ☑ Negative test

Function call analysis

- `IScrollERC721(_token).ownerOf(_tokenIds[i])`
  - **What is controllable?** `_token`, `_tokenIds[i]`.
  - **If return value controllable, how is it used and how can it go wrong?** The owner of the token.
  - **What happens if it reverts, reenters, or does other unusual control flow?** N/A.
- `IScrollERC721(_token).burn(_tokenIds[i])`
- **What is controllable?** `_token`, `_tokenIds[i]`.
  - **If return value controllable, how is it used and how can it go wrong?** N/A.
  - **What happens if it reverts, reenters, or does other unusual control flow?** Means user does not actually own the token; function reverts.
- `IL2ScrollMessenger(messenger).sendMessage{value: msg.value}(counterpart, msg.value, _message, _gasLimit)`
  - **What is controllable?**: `_gasLimit`.
  - **If return value controllable, how is it used and how can it go wrong?** N/A.
  - **What happens if it reverts, reenters, or does other unusual control flow?** See finding `3.1`.

Function: `_withdrawERC721(address _token, address _to, uint256 _tokenId, uint256 _gasLimit)`

Deposit an ERC721 token to layer 2.

Inputs

- `_token`
  - **Control**: Full control.
  - **Constraints**: N/A.
  - **Impact**: The token to be withdrawn.

- `_to`
  - **Control**: Full control.
  - **Constraints**: N/A.
  - **Impact**: The recipient of the token on L1.
- `_tokenId`
  - **Control**: Full control.
  - **Constraints**: N/A.
  - **Impact**: The token ID to be withdrawn.
- `_gasLimit`
  - **Control**: Full control.
  - **Constraints**: N/A.
  - **Impact**: The gas limit to be used for the withdrawal on L1.

Branches and code coverage (including function calls)

Intended branches

- The `tokenID` of the token is transferred from the `msg.sender` to this contract on L1.
  ☐ Test coverage
- Assure the `_token` is a valid one, and cannot overwrite the mapping in anyway.
  ☑ Test coverage
- The `finalizeWithdrawERC721` function is called on the `L1ERC721Gateway` contract.
  ☐ Test coverage
- The `sendMessage` function is called on the L2ScrollMessenger contract.
  ☐ Test coverage
- Ensure the minted L2 Token is burned.
  ☑ Test coverage

Negative behavior

- Should technically not be callable on the same token twice, unless it has been transferred back from L2.
  ☐ Negative test
- Should not allow arbitrary tokens to be deposited for legitimate l2 tokens.
  ☑ Negative test

Function call analysis

- `IScrollERC721(_token).ownerOf(_tokenId)`
  - **What is controllable?** `tokenId`.
  - **If return value controllable, how is it used and how can it go wrong?** Is the

withdrawer, the owner.
  - **What happens if it reverts, reenters, or does other unusual control flow?** N/A.
- `IScrollERC721(_token).burn(_tokenId)`
  - **What is controllable?** `_tokenId`, only if owner.
  - **If return value controllable:** Discarded.
  - **What happens if it reverts, reenters, or does other unusual control flow?** N/A.
- `IL2ScrollMessenger(messenger).sendMessage(counterpart, msg.value, _message, _gasLimit)`
  - **What is controllable?** `_gasLimit`.
  - **If return value controllable, how is it used and how can it go wrong?** Discarded.
  - **What happens if it reverts, reenters, or does other unusual control flow?** N/A. Cross-chain call not successful; covered in finding 3.1.

5.12 Module: L2ERC1155Gateway.sol

Function: `finalizeBatchDepositERC1155(address _l1Token, address _l2Token, address _from, address _to, uint256[] _tokenIds, uint256[] _amounts)`

Allow the batch deposit of multiple `tokenIds` and `amounts` of the same ERC1155 token from L1 to L2.

Inputs

- `_l1Token`
  - **Control**: Fully controlled by the counterpart.
  - **Constraints**: N/A.
  - **Impact**: l1 token deposited.
- `_l2Token`
  - **Control**: Fully controlled by the counterpart.
  - **Constraints**: N/A.
  - **Impact**: l2 token to be minted.
- `_from`
  - **Control**: Fully controlled by the counterpart.
  - **Constraints**: N/A.
  - **Impact**: The sender of the message on L2.
- `_to`

  - **Control**: Fully controlled by the counterpart.
  - **Constraints**: N/A.
  - **Impact**: The recipient of the token on L1.
- `_tokenIds`
  - **Control**: Fully controlled by the counterpart.
  - **Constraints**: N/A.
  - **Impact**: The token IDs to be deposited.
- `_amounts`
  - **Control**: Fully controlled by the counterpart.
  - **Constraints**: N/A.
  - **Impact**: The amounts of tokens to be deposited.

Branches and code coverage (including function calls)

Intended branches

- Mint `_amounts` tokens to the receipient
  ☑ Test coverage
- Check that the lengths of the `_tokenIds` and `_amounts` arrays are the same
  ☑ Test coverage

Negative behavior

- Should not be callable by anyone other than the counterpart.
  ☑ Negative test

Function call analysis

- `IScrollERC1155(_l2Token).batchMint(_to, _tokenIds, _amounts, "")`
  - **What is controllable?** `_to`, `_tokenIds`, `_amounts`.
  - **If return value controllable, how is it used and how can it go wrong?** N/A.
  - **What happens if it reverts, reenters, or does other unusual control flow?** N/A.

Function: `finalizeDepositERC1155(address _l1Token, address _l2Token, address _from, address _to, uint256 _tokenId, uint256 _amount)`

Allow the deposit of an ERC1155 token from L1 to L2.

Inputs

- `_l1Token`
  - **Control**: Fully controlled by the counterpart.

  - **Constraints**: None.
  - **Impact**: The token that was deposited on L1.
- `_l2Token`
  - **Control**: Fully controlled by the counterpart.
  - **Constraints**: None.
  - **Impact**: Token to be minted.
- `_from`
  - **Control**: Fully controlled by the counterpart.
  - **Constraints**: N/A.
  - **Impact**: The sender of the message on L1.
- `_to`
  - **Control**: Fully controlled by the counterpart.
  - **Constraints**: N/A.
  - **Impact**: The recipient of the token on L2.
- `_tokenId`
  - **Control**: Fully controlled by the counterpart.
  - **Constraints**: N/A.
  - **Impact**: The `tokenId`.
- `_amount`
  - **Control**: Fully controlled by the counterpart.
  - **Constraints**: N/A.
  - **Impact**: The amount of tokens to be deposited.

Branches and code coverage (including function calls)

Intended branches

- Mints the appropriate amount of L2 token
  ☐ Test coverage
- Decreases the balance of the `address(this)` on L1 with the `tokenId` by the `amount`
  ☑ Test coverage

Negative behavior

- Should not be callable by anyone other than the counterpart.
  ☑ Negative test
- Should not be callable if the `_l2Token` does not match the `tokenMapping[_l1Token]`
  ☐ Negative test
- Should not allow calling if `l2Token == 0`.
  ☐ Negative test

Function call analysis

- `IScrollERC1155(_l2Token).mint(_to, _tokenId, _amount, "")`
  - **What is controllable?** `_to`, `_tokenId`, `_amount` (deposited on L1).
  - **If return value controllable, how is it used and how can it go wrong?** N/A.
  - **What happens if it reverts, reenters, or does other unusual control flow?** N/A.

Function: `initialize(address _counterpart, address _messenger)`

Initialize the storage variables as well as the inherited contracts'.

Inputs

- `_counterpart`
  - **Control**: Fully controlled.
  - **Constraints**: N/A.
  - **Impact**: The address of the counterpart.
- `_messenger`
  - **Control**: Fully controlled.
  - **Constraints**: N/A.
  - **Impact**: Scroll messenger address.

Branches and code coverage (including function calls)

Intended branches

- Should call the `OwnableUpgradeable.__Ownable_init()` function.
  ☑ Test coverage
- Should call the `ScrollGatewayBase._initialize()` function.
  ☑ Test coverage
- Should call the `ERC1155HolderUpgradeable.__ERC1155Holder_init()` function.
  ☐ Test coverage
- Should set the counterpart to `_counterpart`.
  ☑ Test coverage
- Should set the messenger to `_messenger`.
  ☑ Test coverage

Negative behavior

- Should not be callable twice.
  ☑ Negative test

Function: `updateTokenMapping(address _l1Token, address _l2Token)`

Update the mapping of layer 1 to layer 2 token.

Inputs

- `_l1Token`
  - **Control**: Full control
  - **Constraints**: n/a
  - **Impact**: The layer 1 token to be approved
- `_l2Token`
  - **Control**: Full control
  - **Constraints**: checked that `_l2Token` $\neq$ `address(0)`
  - **Impact**: The layer 2 token to be approved

Branches and code coverage (including function calls)

Intended branches

- Assumed this should not change that often, or at least not during important bridge operations
  - ☐ Test coverage
- Update the state of the `tokenMapping` to reflect the addition.
  - ☑ Test coverage

Negative behaviour

- Should not be callable by anyone other than the `owner`
  - ☑ Negative test
- Should not allow double mapping of tokens
  - ☐ Negative test

Function: `_batchWithdrawERC1155(address _token, address _to, uint256[] calldata _tokenIds, uint256[] calldata _amounts, uint256 _gasLimit)`

Batch withdrawal of L1 tokens from the L2.

Inputs

- `_token`
  - **Control**: Full control.
  - **Constraints**: N/A.
  - **Impact**: The token to be withdrawn.

- `_to`
  - **Control**: Full control.
  - **Constraints**: N/A.
  - **Impact**: The recipient of the token on L2.
- `_tokenIds`
  - **Control**: Full control.
  - **Constraints**: Checked the length of `_tokenIds` and `_amounts` are the same.
  - **Impact**: The token IDs to be withdrawn.
- `_amounts`
  - **Control**: Full control.
  - **Constraints**: Checked amount of tokens to be withdrawn to be greater than 0 - also that the length of `_tokenIds` and `_amounts` are the same.
  - **Impact**: The amounts of tokens to be withdrawn.
- `_gasLimit`
  - **Control**: Full control.
  - **Constraints**: N/A.
  - **Impact**: The gas limit to be used for the withdraw on L2.

Branches and code coverage (including function calls)

Intended branches

- The `tokenIDs` are burned
  ☑ Test coverage
- Assure the `_token` is a valid one, and cannot overwrite the mapping in anyway.
  ☑ Test coverage
- The `finalizeBatchWithdrawERC1155` function is called on the `L1ERC1155Gateway` contract.
  ☐ Test coverage
- The `sendMessage` function is called on the L2ScrollMessenger contract.
  ☐ Test coverage
- Assure that the lengths of `_tokenIds` and `_amounts` are the same.
  ☑ Test coverage

Negative behavior

- Should not allow arbitrary tokens to be withdrawn for legitimate l1 tokens.
  ☑ Negative test
- Should not allow user to withdraw more than they have.
  ☑ Negative test

Function call analysis

- `IScrollERC1155(_token).batchBurn(msg.sender, _tokenIds, _amounts)`
  - **What is controllable?** `_tokenIds`, `_amounts`.
  - **If return value controllable, how is it used and how can it go wrong?** N/A.
  - **What happens if it reverts, reenters, or does other unusual control flow?** N/A.
- `IL2ScrollMessenger(messenger).sendMessage{value: msg.value}(counterpart, msg.value, _message, _gasLimit)`
  - **What is controllable?** `_gasLimit`.
  - **If return value controllable, how is it used and how can it go wrong?** N/A.
  - **What happens if it reverts, reenters, or does other unusual control flow?** N/A.

Function: `_withdrawERC1155(address _token, address _to, uint256 _tokenId, uint256 _amount, uint256 _gasLimit)`

Facilitates the withdraw of ERC1155 tokens to L2.

Inputs

- `_token`
  - **Control**: Full control.
  - **Constraints**: `tokenMapping[_token]` $\neq$ `address(0)`.
  - **Impact**: The token to be withdrawn.
- `_to`
  - **Control**: Full control.
  - **Constraints**: N/A.
  - **Impact**: The recipient of the token on L1.
- `_tokenId`
  - **Control**: Full control.
  - **Constraints**: N/A.
  - **Impact**: The token ID to be withdrawn.
- `_amount`
  - **Control**: Full control.
  - **Constraints**: Checked amount is not zero.
  - **Impact**: The amount of tokens to be withdrawn.
- `_gasLimit`
  - **Control**: Full control.
  - **Constraints**: N/A.

- **Impact**: The gas limit to be used for the withdrawal on L1.

Branches and code coverage (including function calls)

Intended branches

- The tokenId is burned from the msg.sender
  ☑ Test coverage
- The correct amount of tokenId is burned
  ☑ Test coverage
- Assure the `_token` is a valid one, and cannot overwrite the mapping in anyway.
  ☑ Test coverage
- The `finalizeWithdrawERC1155` function is called on the `L1ERC1155Gateway` contract.
  ☐ Test coverage
- The `sendMessage` function is called on the L2ScrollMessenger contract.
  ☐ Test coverage

Negative behavior

- Should not allow arbitrary tokens to be withdrawn for legitimate l1 tokens.
  ☐ Negative test
- Should not allow user to withdraw more than they have.
  ☑ Negative test

Function call analysis

- `IScrollERC1155(_token).burn(msg.sender, _tokenId, _amount)`
- **What is controllable?** `_tokenId`, `_amount`.
  - **If return value controllable, how is it used and how can it go wrong?** N/A.
  - **What happens if it reverts, reenters, or does other unusual control flow?** N/A (no tokens to burn).
- `IL2ScrollMessenger(messenger).sendMessage(counterpart, msg.value, _message, _gasLimit)`
  - **What is controllable?** `_gasLimit`.
  - **If return value controllable, how is it used and how can it go wrong?** N/A.
  - **What happens if it reverts, reenters, or does other unusual control flow?** N/A

5.13 Module: L2ETHGateway.sol

Function: `finalizeDepositETH(address _from, address _to, uint256 _amount, byte[] _data)`

Finalizes the deposit of ETH from L1 to L2. This function is called by the counterpart.

Inputs

- `_from`
  - **Control**: Full control.
  - **Constraints**: N/A.
  - **Impact**: Source of ETH from L1.
- `_to`
  - **Control**: Full control.
  - **Constraints**: N/A.
  - **Impact**: Destination of ETH.
- `_amount`
  - **Control**: Full control.
  - **Constraints**: N/A.
  - **Impact**: Amount of ETH to deposit.
- `_data`
  - **Control**: Full control.
  - **Constraints**: No constraints here. This is critical since it is important that it properly defends against arbitrary calls! Important to use an interface here, so safeguard against arbitrary contract type calls.
  - **Impact**: Calldata to be passed to `_to`.

Branches and code coverage (including function calls)

Intended branches

- Increase the balance of `_to` by `_amount`
  ☑ Test coverage
- Decrease the balance of `_from` by `_amount`
  ☐ Test coverage
- Pass the call to `_to` with `_data` as calldata.
  ☐ Test coverage

Negative behaviour

Function call analysis

- `_to.call{value: _amount}("")`

- **What is controllable?** `_to` and `_amount` - also the calldata `_data`, but this is not used at the moment.
- **If return value controllable, how is it used and how can it go wrong?** Return value tells whether the call succeeded or not. If it fails, the ETH is not transferred.
- **What happens if it reverts, reenters, or does other unusual control flow?** If it reverts, the ETH is not transferred. It can reenter and use the `L1ETHGateway` as a proxy to transfer ETH to any address. At the moment, this is not an issue, since calldata is not used, but this should be fixed in the future by using an interface.

Function: `initialize(address _counterpart, address _router, address _messenger)`

Initialize the storage variables as well as the inherited contracts'.

Inputs

- `_counterpart`
  - **Control**: Fully controlled.
  - **Constraints**: N/A.
  - **Impact**: The address of the counterpart.
- `_messenger`
  - **Control**: Fully controlled.
  - **Constraints**: N/A.
  - **Impact**: Scroll messenger address.
- `_router`
  - **Control**: Fully controlled.
  - **Constraints**: N/A.
  - **Impact**: Router address.

Branches and code coverage (including function calls)

Intended branches

- Should call the `ScrollGatewayBase._initialize()` function.
  ☑ Test coverage
- Should set the counterpart to `_counterpart`.
  ☑ Test coverage
- Should set the messenger to `_messenger`.
  ☑ Test coverage

- Should set the router to `_router`.
  ☑ Test coverage

Negative behavior

- Should not be callable twice.
  ☑ Negative test

Function: `_withdraw(address _to, uint256 _amount, byte[] _data, uint256 _gasLimit)`

Facilitate the withdrawal of native tokens from L1 to L2.

Inputs

- `_to`
  - **Control**: Full control.
  - **Constraints**: N/A.
  - **Impact**: The destination address for the withdraw.
- `_amount`
  - **Control**: Full control.
  - **Constraints**: N/A.
  - **Impact**: The amount of native tokens to be withdrawn.
- `_data`
  - **Control**: Full control.
  - **Constraints**: N/A.
  - **Impact**: The data to be passed to the recipient.
- `_gasLimit`
  - **Control**: Full control.
  - **Constraints**: N/A.
  - **Impact**: The gas limit for the withdraw.

Branches and code coverage (including function calls)

Intended branches

- Should forward the payload on to the messenger.
  ☐ Test coverage
- Decrease the balance of `msg.sender` by `_amount`.
  ☑ Test coverage
- Increase the balance of `messenger` by `_amount`.
  ☑ Test coverage

Negative behaviour

- Should not allow `_amount` to be ≥ `msg.value`.
  ☑ Negative test

Function call analysis

- `IL2ScrollMessenger(messenger).sendMessage{value: msg.value}(counterpart, _amount, _message, _gasLimit)`
  - **What is controllable?** `_message`, `_gasLimit`.
  - **If return value controllable, how is it used and how can it go wrong?** N/A.
  - **What happens if it reverts, reenters, or does other unusual control flow?** N/A.

5.14 Module: L2GatewayRouter.sol

Function: `initialize(address _ethGateway, address _defaultERC20Gateway)`

Initialize the storage variables as well as the inherited contracts'.

Inputs

- `_ethGateway`
- **Control**: Full control.
  - **Constraints**: Checked for nonzero.
  - **Impact**: Sets the `ethGateway` variable.
- `_defaultERC20Gateway`
  - **Control**: Full control.
  - **Constraints**: Checked for nonzero.
  - **Impact**: Sets the `defaultERC20Gateway` variable.

Branches and code coverage (including function calls)

Intended branches

- Should set the default ERC20 gateway.
  ☑ Test coverage
- Should set the ETH gateway.
  ☑ Test coverage
- Call to `OwnableUpgradeable.__Ownable_init()` should succeed.
  ☑ Test coverage

Negative behavior

- Should not be callable twice.
  ☑ Negative test

Function: `setDefaultERC20Gateway(address _defaultERC20Gateway)`

Sets the address of the default ERC20 gateway contract.

Inputs

- `_defaultERC20Gateway`
  - **Control**: Full control.
  - **Constraints**: None.
  - **Impact**: The address of the default ERC20 gateway contract.

Branches and code coverage (including function calls)

Intended branches

- Should update the state of the `defaultERC20Gateway` variable.
  ☑ Test coverage
- Assumes the contract will not be left in an inconsistent state after the change.
  ☐ Test coverage

Negative behavior

- Should not be callable by anyone other than the owner.
  ☑ Negative test

Function: `setERC20Gateway(address[] _tokens, address[] _gateways)`

Updates the mapping from token address to gateway address.

Inputs

- `_tokens`
  - **Control**: Full control.
  - **Constraints**: Length of `_tokens` must match length of `_gateways`.
  - **Impact**: The list of addresses of tokens to update.
- `_gateways`
  - **Control**: Full control.
  - **Constraints**: Length of `_gateways` must match length of `_tokens`.
  - **Impact**: The list of addresses of gateways that will be accesible.

Branches and code coverage (including function calls)

Intended branches

- Should update the state of the `ERC20Gateway` mapping for each token in `_tokens` with the corresponding gateway in `_gateways`.
  ☑ Test coverage
- Assumes the contract will not be left in an inconsistent state after the change.
  ☐ Test coverage

Negative behavior

- Should only be callable by the owner.
  ☑ Negative test

Function: `setETHGateway(address _ethGateway)`

Sets the address of the ETH gateway contract.

Inputs

- `_ethGateway`
  - **Control**: Full control.
  - **Constraints**: None.
  - **Impact**: The address of the ETH gateway contract.

Branches and code coverage (including function calls)

Intended branches

- Should update the state of the `ethGateway` variable.
  ☑ Test coverage
- Assume the contract will not be left in an inconsistent state after the change.
  ☐ Test coverage

Negative behavior

- Should not be callable by anyone other than the owner.
  ☑ Negative test

Function: `withdrawETHAndCall(address _to, uint256 _amount, byte[] _data , uint256 _gasLimit)`

Allows user to withdraw ETH and call a contract with the withdrawn ETH on L1.

Inputs

- `_to`
  - **Control**: Full control.
  - **Constraints**: N/A.
  - **Impact**: The destination address for the withdrawal.
- `_amount`
  - **Control**: Full control.
  - **Constraints**: Should check that `msg.value ≥ _amount`.
  - **Impact**: The amount of the ERC20 native token to be transferred.
- `_data`
  - **Control**: Full control.
  - **Constraints**: N/A.
  - **Impact**: The data to be passed to the recipient.
- `_gasLimit`
  - **Control**: Full control.
  - **Constraints**: N/A.
  - **Impact**: The gas limit for the withdrawal.

Branches and code coverage (including function calls)

Intended branches

- Allow user to withdraw ETH and call a contract with the withdrawn ETH on the L1.
  - ☐ Test coverage
- Decrease the balance of ETH for the user.
  - ☐ Test coverage
- Increase the balance of ETH for the user on the L1.
  - ☐ Test coverage
- Increase the balance of the messenger contract.
  - ☐ Test coverage

Negative behavior

- Should not allow calling with `msg.value < amount`. This in fact should be checked at the level of `sendMessage` in `L2ETHGateway`.
  - ☐ Negative test
- Should not allow calls to arbitrary addresses on the L1 side; this has to be enforced project wide.
  - ☐ Negative test

Function call analysis

- `IL2ETHGateway(_gateway).withdrawETHAndCall{value: msg.value}(_to, _amount, _routerData, _gasLimit)`
  - **What is controllable?** `msg.value`, `_to`, `_amount`.
  - **If return value controllable, how is it used and how can it go wrong?** N/A.
  - **What happens if it reverts, reenters, or does other unusual control flow?** N/A.

5.15 Module: L2ScrollMessenger.sol

Function: `initialize(address _counterpart, address _feeVault, address _rollup, address _messageQueue)`

Initializes the storage of L1ScrollMessenger.

Inputs

- `_counterpart`
  - **Control**: Full control.
  - **Constraints**: None.
  - **Impact**: The address of counterpart.
- `_feeVault`
  - **Control**: Full control.
  - **Constraints**: None.
  - **Impact**: The address of the vault where fees are sent.
- `_rollup`
  - **Control**: Full control.
  - **Constraints**: None.
  - **Impact**: The address of ScrollChain contract.
- `_messageQueue`
  - **Control**: Full control.
  - **Constraints**: None.
  - **Impact**: Address of message queue.

Branches and code coverage (including function calls)

Intended branches

- Set all important variables.
  - ☐ Test coverage

- Call all underlying initializer functions.
  - ☐ Test coverage

Negative behavior

- Should not be callable multiple times.
  - ☐ Negative test

Function retryMessageWithProof(address _from, address _to, uint256 _value, uint256 _nonce, bytes memory _message, L1MessageProof calldata _proof)

Allow sending messages from L1 to L2.

Inputs

- `_from`
  - **Control**: Full control.
  - **Constraints**: Must be in proof.
  - **Impact**: Source.
- `_to`
  - **Control**: Full control.
  - **Constraints**: Must be in proof.
  - **Impact**: Destination.
- `_value`
  - **Control**: Full control.
  - **Constraints**: Must be in proof.
  - **Impact**: Amount.
- `_nonce`
  - **Control**: Full control.
  - **Constraints**: Must be in proof.
  - **Impact**: Nonce.
- `_message`
  - **Control**: Full control.
  - **Constraints**: Must be in proof.
  - **Impact**: Calldata to be passed on.
- `_proof`
  - **Control**: Full control.
  - **Constraints**: Must be in proof.
  - **Impact**: Proof to be checked against.

Branches and code coverage (including function calls)

Intended branches

- Message was relayed at least once.
  - ☐ Test coverage
- Reentrancy check/XDomain check.
  - ☐ Test coverage

Negative behavior

- Cannot supply the same proof twice.
  - ☐ Test coverage

Function call analysis

- `verifyMessageInclusionStatus(_proof.blockHash, _xDomainCalldataHash, _proof.stateRootProof)`
  - **What is controllable?** `_xDomainCalldataHash`.
  - **If return value controllable, how is it used and how can it go wrong?** Is the proof valid?
  - **What happens if it reverts, reenters, or does other unusual control flow?** N/A.
- `_executeMessage(_from, _to, _value, _message, _xDomainCalldataHash)`
  - **What is controllable?** Everything.
  - **If return value controllable, how is it used and how can it go wrong?** N/A.
  - **What happens if it reverts, reenters, or does other unusual control flow?** N/A.

Function: `sendMessage(address _to, uint256 _value, byte[] _message, uint256 _gasLimit, address _refundAddress)`

Allow sending messages from L1 to L2.

Inputs

- `_to`
  - **Control**: Full control.
  - **Constraints**: None.
  - **Impact**: The destination address of the message on L1.
- `_value`
  - **Control**: Full control.
  - **Constraints**: Checked against `msg.value` + estimated fee.

  - **Impact**: The value to be sent to the destination address on L1.
- `_message`
  - **Control**: Full control.
  - **Constraints**: None.
  - **Impact**: The payload of the message.
- `_gasLimit`
  - **Control**: Full control; used in calculation of the fee.
  - **Constraints**: None.
  - **Impact**: The gas limit for the message on L2.

Branches and code coverage (including function calls)

Intended branches

- Should deplete the caller's balance by the amount that is sent to the destination address on L1.
  ☐ Test coverage
- Should refund any excess ETH to the `_refundAddress`.
  ☑ Test coverage
- Should add the crafted message to the queue.
  ☑ Test coverage
- Should emit the `SentMessage` event.
  ☑ Test coverage
- Ensure that `msg.value ≥ _value + fee`.
  ☑ Test coverage

Negative behavior

- Should not be callable when paused.
  ☐ Negative test
- Should not be able to craft the same message twice. Here the `messageNonce` is used as a unique identifier for the message as well as the `isL2MessageSent` mapping.
  ☑ Negative test

Function call analysis

- `IL1GasPriceOracle(gasOracle).l1BaseFee()`
  - **What is controllable?** N/A.
  - **If return value controllable, how is it used and how can it go wrong?** Fee to be multiplied by the gas limit.
  - **What happens if it reverts, reenters, or does other unusual control flow?**

N/A.

- `IL1MessageQueue(_messageQueue).nextMessageIndex()`
  - **What is controllable?** N/A.
  - **If return value controllable, how is it used and how can it go wrong?** Computes the nonce according to the message queue.
  - **What happens if it reverts, reenters, or does other unusual control flow?** N/A.
- `feeVault.call{value: _fee}("")`
  - **What is controllable?** N/A.
  - **If return value controllable, how is it used and how can it go wrong?** Sends the fee to the fee vault.
  - **What happens if it reverts, reenters, or does other unusual control flow?** N/A.
- `L2MessageQueue(messageQueue).appendMessage(_xDomainCalldataHash)`
  - **What is controllable?** `gasLimit`.
  - **If return value controllable, how is it used and how can it go wrong?** Adds the message to the queue.
  - **What happens if it reverts, reenters, or does other unusual control flow?** N/A.

Function: `setPause(bool _status)`

Pause or unpause the contract.

Inputs

- `_status`
  - **Control**: Full control.
  - **Constraints**: N/A.
  - **Impact**: Dictates the pause status to update.

Branches and code coverage (including function calls)

Intended branches

- Should update the `paused` status.
  ☑ Test coverage

Negative behavior

- Should not be called by anyone other than the `owner`.
  ☑ Negative test

Function `_executeMessage( address _from, address _to, uint256 _value, bytes memory _message, bytes32 _xDomainCalldataHash)`

Executes the message invoking the callback.

Inputs

- `_from`
  - **Control**: Full control.
  - **Constraints**: N/A.
  - **Impact**: Source.
- `_to`
  - **Control**: Full control.
  - **Constraints**: `to ≠ MessageQueue || to ≠ address(this)`.
  - **Impact**: Destination.
- `_value`
  - **Control**: Full control.
  - **Constraints**: N/A.
  - **Impact**: Amount.
- `_message`
  - **Control**: Full control.
  - **Constraints**: N/A.
  - **Impact**: Calldata.
- `_xDomainCalldataHash`
  - **Control**: Full control.
  - **Constraints**: N/A.
  - **Impact**: Hash of the message.

Branches and code coverage (including function calls)

- Calls the callback function.
  ☑ Test Coverage

Negative behavior

Negative behaviour

- Does not allow the `_to` to be MessageQueue or the ScrollMessenger
  ☑ Test coverage

Function call analysis

- `_to.call{value: _value}(_message)`
  - **What is controllable?** `_to` (partial), `_message`.
  - **If return value controllable, how is it used and how can it go wrong?** Determines if the call is put into the retry queue.
  - **What happens if it reverts, reenters, or does other unusual control flow?** N/A.

5.16 Module: L2StandardERC20Gateway.sol

Function: `finalizeDepositERC20(address _l1Token, address _l2Token, address _from, address _to, uint256 _amount, byte[] _data)`

Perform the finalization of deposit ERC20 token from L1 to L2.

Inputs

- `_l1Token`
  - **Control**: Full control.
  - **Constraints**: Computes the L2TokenAddress and checks if it exists.
  - **Impact**: l1Token that was deposited.
- `_l2Token`
  - **Control**: Full control.
  - **Constraints**: N/A.
  - **Impact**: l2Token to be minted.
- `_from`
  - **Control**: Full control.
  - **Constraints**: N/A (checked in l2).
  - **Impact**: Sender from l1.
- `_to`
  - **Control**: Full control.
  - **Constraints**: N/A.
  - **Impact**: Destination address.
- `_amount`
  - **Control**: Full control.
  - **Constraints**: N/A (checked in l2).
  - **Impact**: Amount to be withdrawn.
- `_data`
  - **Control**: Full control.

- **Constraints**: N/A (checked in l2).
- **Impact**: Data to be passed on to the recipient.

Branches and code coverage (including function calls)

Intended branches

- Mint correct amount of tokens
  ☑ Test coverage
- calculate the correct L2 Token from the L1 token
  ☑ Test coverage
- Forward the `_data` to the `_to` on layer 2
  ☐ Test coverage

Negative behaviour

- Should not be callable by anyone other than the `counterpart`
  ☑ Negative test
- Should not be callable multiple times for the same deposited token
  ☑ Negative test
- Should not be callable with a zero `_amount`
  ☑ Negative test

Function call analysis

- `IScrollStandardERC20Factory(tokenFactory).computeL2TokenAddress(address(this), _l1Token)`
- **What is controllable?** `_l1Token`.
  - **If return value controllable, how is it used and how can it go wrong?** Not controllable - the deterministic hash according to the clones library on L2.
  - **What happens if it reverts, reenters, or does other unusual control flow?** N/A.
- `_l2Token.isContract()`
  - **What is controllable?** `_l2Token`.
  - **If return value controllable, how is it used and how can it go wrong?** Check if the token is a contract.
  - **What happens if it reverts, reenters, or does other unusual control flow?** N/A.

Function: `initialize(address _counterpart, address _router, address _messenger, address _l2TokenImplementation, address _l2TokenFactory)`

Initialize the storage of L1StandardERC20Gateway.

Inputs

- `_counterpart`
  - **Control**: Full control.
  - **Constraints**: N/A.
  - **Impact**: The address of the counterpart.
- `_router`
  - **Control**: Full control.
  - **Constraints**: N/A.
  - **Impact**: The address of the router.
- `_messenger`
  - **Control**: Full control.
  - **Constraints**: N/A.
  - **Impact**: The address of the messenger.
- `_l2TokenImplementation`
  - **Control**: Full control.
  - **Constraints**: N/A.
  - **Impact**: The address of the l2TokenImplementation.
- `_l2TokenFactory`
  - **Control**: Full control.
  - **Constraints**: N/A.
  - **Impact**: The address of the l2TokenFactory.

Branches and code coverage (including function calls)

Intended branches

- Set the counterpart address.
  ☑ Test coverage
- Set the router address.
  ☑ Test coverage
- Set the messenger address.
  ☑ Test coverage
- Set the l2TokenImplementation address.
  ☑ Test coverage
- Set the l2TokenFactory address.

☑ Test coverage

- Call the initialize function of the base contract.
  ☑ Test coverage

Negative behavior

- Should not be callable twice.
  ☑ Negative test

Function: `_withdraw(address _token, address _to, uint256 _amount, byte[] _data, uint256 _gasLimit)`

Facilitate the withdraw and transfer of an ERC20 token from layer 1 to layer 2.

Inputs

- `_token`
- **Control**: Full control.
  - **Constraints**: Checked `tokenMapping[_token]` existence.
  - **Impact**: The ERC20 token to be withdrawn.
- `_to`
- **Control**: Full control.
  - **Constraints**: N/A.
  - **Impact**: The destination address for the withdraw.
- `_amount`
- **Control**: Full control.
  - **Constraints**: > 0.
  - **Impact**: The amount of the ERC20 token to be withdrawn.
- `_data`
- **Control**: Full control.
  - **Constraints**: N/A.
  - **Impact**: The data to be passed to the recipient.
- `_gasLimit`
  - **Control**: Full control.
  - **Constraints**: N/A.
  - **Impact**: The gas limit for the withdraw.

Branches and code coverage (including function calls)

Intended branches

- Ensure the difference of the after and before balance is positive.
  ☐ Test coverage
- Should burn `_token` by `_amount` for the `msg.sender`
  ☑ Test coverage
- Should increase the balance of the recipient on L1
  ☐ Test coverage
- Should account for router data
  ☑ Test coverage
- Should forward the payload on to the messenger.
  ☐ Test coverage

Negative behavior

- Should not permit withdrawing fictitious tokens.
  ☑ Negative test
- Should not permit withdrawing zero amount.
  ☑ Negative test
- Should ensure that user withdraws the exact amount of token.
  ☑ Negative test

Function call analysis

- `IScrollStandardERC20(_token).burn(_from, _amount)`
- **What is controllable?** `_token` (partial), `_amount`.
  - **If return value controllable, how is it used and how can it go wrong?** N/A.
  - **What happens if it reverts, reenters, or does other unusual control flow?** N/A.
- `IL2ScrollMessenger(messenger).sendMessage{value: msg.value}(counterpart, 0, _message, _gasLimit)`
  - **What is controllable?** `_message`(partially), `_gasLimit`.
  - **If return value controllable, how is it used and how can it go wrong?** N/A.
  - **What happens if it reverts, reenters, or does other unusual control flow?** N/A.

5.17 Module: L2WETHGateway.sol

Function: `finalizeDepositERC20(address _l1Token, address _l2Token, address _from, address _to, uint256 _amount, byte[] _data)`

Allows the finalization of a deposit from L1 to L2.

Inputs

- `_l1Token`
  - **Control**: Full control.
  - **Constraints**: `l1Token == l1WETH`.
  - **Impact**: L1WETH.
- `_l2Token`
  - **Control**: Full control.
  - **Constraints**: `== WETH`.
  - **Impact**: L2WETH.
- `_from`
  - **Control**: Full control.
  - **Constraints**: N/A (checked in l2).
  - **Impact**: Sender from l1.
- `_to`
  - **Control**: Full control.
  - **Constraints**: N/A.
  - **Impact**: Destination address.
- `_amount`
  - **Control**: Full control.
  - **Constraints**: Checked that it is `msg.value`.
  - **Impact**: Amount to be deposited.
- `_data`
  - **Control**: Full control.
  - **Constraints**: N/A (checked in l2).
  - **Impact**: Data to be passed on to the recipient.

Branches and code coverage (including function calls)

Intended branches

- `msg.value` should be equal to `_amount`
  - ☐ Test coverage
- `_l1Token` should be equal to `WETH`
  - ☑ Test coverage
- `_l2Token` should be equal to `l2WETH`
  - ☑ Test coverage
- `_amount` should be greater than zero
  - ☑ Test coverage
- Should forward `_data` to `_to`. Maybe there could be an interface for this, such

that it's safer during the call.
  ☐ Test coverage

Negative behaviour

- Should not be callable by anyone other than the `counterpart`
  ☑ Negative test
- Should not be callable multiple times for the same deposit action in L1
  ☑ Negative test
- Should not be callable with a zero `_amount`
  ☑ Negative test

Function call analysis

- `IWETH(_l2Token).deposit{value: _amount}();`
- **What is controllable?** `_l1Token`, `_amount`.
  - **If return value controllable, how is it used and how can it go wrong?** N/A.
  - **What happens if it reverts, reenters, or does other unusual control flow?** N/A.
- `IERC20(_l2Token).safeTransfer(_to, _amount);`
  - **What is controllable?** `_l1Token`, `_to`, `_amount`.
  - **If return value controllable, how is it used and how can it go wrong?** N/A.
  - **What happens if it reverts, reenters, or does other unusual control flow?** N/A.

Function: `initialize(address _counterpart, address _router, address _messenger)`

Initialize the storage variables as well as the inherited contracts'.

Inputs

- `_counterpart`
  - **Control**: Fully controlled.
  - **Constraints**: N/A.
  - **Impact**: The address of the counterpart.
- `_messenger`
  - **Control**: Fully controlled.
  - **Constraints**: N/A.
  - **Impact**: Scroll messenger address.
- `_router`

- **Control**: Fully controlled.
- **Constraints**: N/A.
- **Impact**: Router address.

Branches and code coverage (including function calls)

Intended branches

- Should call the `ScrollGatewayBase._initialize()` function.
  ☑ Test coverage
- Should set the counterpart to `_counterpart`.
  ☑ Test coverage
- Should set the messenger to `_messenger`.
  ☑ Test coverage
- Should set the router to `_router`.
  ☑ Test coverage

Negative behavior

- Should not be callable twice.
  ☑ Negative test

Function: `receive()`

Receive native Ether from WETH contract.

Branches and code coverage (including function calls)

Intended branches

- Should technically only be called by WETH during unwrap.
  ☐ Test coverage

Negative behavior

- Should not assume that the contract cannot receive Ether in another way (e.g., selfdestruct).
  ☐ Negative test

Function: `_withdraw(address _token, address _to, uint256 _amount, byte[] _data, uint256 _gasLimit)`

Facilitate the withdraw and transfer of WETH from layer 1 to layer 2.

Inputs

- `_token`
- **Control**: Full control.
  - **Constraints**: Checked it is the WETH address.
  - **Impact**: The ERC20 token to be withdrawn.
- `_to`
- **Control**: Full control.
  - **Constraints**: N/A.
  - **Impact**: The destination address for the withdraw.
- `_amount`
- **Control**: Full control.
  - **Constraints**: `_amount > 0`.
  - **Impact**: The amount of the ERC20 token to be withdrawn.
- `_data`
- **Control**: Full control.
  - **Constraints**: N/A.
  - **Impact**: The data to be passed to the recipient.
- `_gasLimit`
  - **Control**: Full control.
  - **Constraints**: N/A.
  - **Impact**: The gas limit for the withdraw.

Branches and code coverage (including function calls)

Intended branches

- Should decrease the balance of `WETH` by `_amount` for the `msg.sender`
  ☑ Test coverage
- Should increase the balance of `native` by `_amount` for the `messenger`. This is because `msg.value` will be used as the `fee` for the `messenger`, whereas the `_amount` will be used as the actual amount to be withdrawn.
  ☑ Test coverage
- Should forward the payload on to the messenger.
  ☐ Test coverage

Negative behavior

- Should not permit withdrawing fictitious tokens.
  ☑ Negative test

5.18 Module: ScrollChain.sol

Function: `commitBatches(Batch[] _batches)`

Commits multiple batches of transactions.

Inputs

- `_batches`
  - **Control**: Fully controlled by the sequencer.
  - **Constraints**: None.
  - **Impact**: The batches to commit.

Branches and code coverage (including function calls)

Intended branches

- Should update the last finalized batch hash for each batch.
  ☑ Test coverage
- Assure that `_batch.blocks.length > 0`.
  ☑ Test coverage
- Assure the `prevStateRoot` is the same as the `newStateRoot` in the parent batch.
  ☑ Test coverage
- Make sure the batches have not already been committed via `batches[publicInputHash].newStateRoot == bytes32(0)`.
  ☑ Test coverage
- Should emit the `FinalizeBatch` event.
  ☑ Test coverage
- Assure the batches are not finalized via `batches[lastFinalizedBatchHash].batchIndex ≥ _batch.batchIndex`.
  ☐ Test coverage
- Assure the batches have not expired.
  ☐ Test coverage

Negative behavior

- Should not be callable by anyone other than the sequencer.
  ☑ Negative test

Function: `commitBatch(Batch _batch)`

Commits next batch of transactions.

Inputs

- `_batch`
  - **Control**: Fully controlled by the sequencer.
  - **Constraints**: None.
  - **Impact**: The batch to commit.

Branches and code coverage (including function calls)

Intended branches

- Should update the last finalized batch hash.
  ☑ Test coverage
- Assure that `_batch.blocks.length > 0`.
  ☑ Test coverage
- Assure the `prevStateRoot` is the same as the `newStateRoot` in the parent batch.
  ☑ Test coverage
- Make sure the batch has not already been committed via `batches[publicInputHash].newStateRoot == bytes32(0)`.
  ☑ Test coverage
- Should emit the `FinalizeBatch` event.
  ☑ Test coverage
- Assure the batch is not finalized via `batches[lastFinalizedBatchHash].batchIndex ≥ _batch.batchIndex`.
  ☐ Test coverage
- Assure the batch has not expired.
  ☐ Test coverage

Negative behavior

- Should not be callable by anyone other than the sequencer.
  ☑ Negative test

Function: `importGenesisBatch(Batch _genesisBatch)`

Imports the genesis batch.

Inputs

- `_genesisBatch`
  - **Control**: Full control.
  - **Constraints**: None.
  - **Impact**: The genesis batch.

Branches and code coverage (including function calls)

Intended branches

- Should be callable by anyone.
  - ☐ Test coverage
- Should technically act like an initializer function, in that it should be callable only once.
  - ☐ Test coverage

Negative behavior

- Should only be callable once. This is theoretically enforced in `lastFinalizedBatchHash == bytes32(0)`.
  - ☑ Negative test
- Should revert if the genesis batch has more than one block.
  - ☑ Negative test
- Should revert if the genesis batch has a nonzero `prevStateRoot`.
  - ☑ Negative test
- Should revert if the genesis block has a zero block hash.
  - ☑ Negative test
- Should revert if the genesis block has a nonzero parent block hash.
  - ☑ Negative test

Function: `initialize(address _messageQueue)`

Initializes the state variables of the contract.

Inputs

- `_messageQueue`
  - **Control**: Full control.
  - **Constraints**: None.
  - **Impact**: The address of the message queue.

Branches and code coverage (including function calls)

Intended branches

- Should set the message queue address.
  - ☑ Test coverage
- Should call all underlying initializers.
  - ☑ Test coverage

Negative behavior

- Should revert if the message queue address is zero.
  ☐ Negative test
- Should not be callable twice.
  ☑ Negative test

Function: `revertBatch(byte[32] _batchHash)`

Revert a nonfinalized batch.

Inputs

- `_batchHash`
  - **Control**: The hash of the batch to revert.
  - **Constraints**: Assumes batch exists.
  - **Impact**: The batch to revert.

Branches and code coverage (including function calls)

Intended branches

- Should delete the committed batch.
  ☑ Test coverage
- The batch should be recommittable.
  ☐ Test coverage

Negative behavior

- Should not be callable by anyone than the sequencer.
  ☑ Negative test
- Should not leave contract in an inconsistent state.
  ☑ Negative test
- Should otherwise not be able to revert a finalized batch.
  ☑ Negative test

Function: `updateSequencer(address _account, bool _status)`

Updates the status of the sequencer.

Inputs

- `_account`
  - **Control**: By owner.

  - **Constraints**: None.
  - **Impact**: The account to update.
- `_status`
  - **Control**: By owner.
  - **Constraints**: None.
  - **Impact**: The new status of the account.

Branches and code coverage (including function calls)

Intended branches

- Should update the status of the account with `_status`.
  ☐ Test coverage
- Should emit an `UpdateSequencer` event.
  ☐ Test coverage
- Assumes the `_status` is different currently.
  ☐ Test coverage
- Assumes there is always at least one sequencer and that it would not leave the contract without a sequencer.
  ☐ Test coverage

Negative behavior

- Should not be callable by anyone other than the owner.
  ☐ Negative test

5.19 Module: ScrollMessengerBase.sol

Function: `updateFeeVault(address _newFeeVault)`

Update the fee vault contract address.

Inputs

- `_newFeeVault`
  - **Control**: Only the contract owner can call this function.
  - **Constraints**: Only the contract owner can call this function.
  - **Impact**: Sets the fee vault contract address.

Branches and code coverage (including function calls)

Intended branches

- Should set the `feeVault` variable to `_newFeeVault`.
  - ☐ Test coverage

Negative behavior

- Should revert if `_newFeeVault` is the zero address.
  - ☐ Negative test
- Should revert if `_newFeeVault` is the same as the current `feeVault` address.
  - ☐ Negative test
- Should revert if called by anyone other than the contract owner.
  - ☑ Negative test
- Should not affect outgoing messages or break the bridge in any way.
  - ☑ Negative test

Function: `updateWhitelist(address _newWhitelist)`

Update the whitelist contract address.

Inputs

- `_newWhitelist`
  - **Control**: Only the contract owner can call this function.
  - **Constraints**: Only the contract owner can call this function.
  - **Impact**: Sets the whitelist contract address.

Branches and code coverage (including function calls)

Intended branches

- Should set the `whitelist` variable to `_newWhitelist`.
  - ☐ Test coverage
- Assumes that the `_newWhitelist` address is a `Whitelist` contract.
  - ☐ Test coverage

Negative behavior

- Should revert if `_newWhitelist` is the zero address.
  - ☐ Negative test
- Should revert if `_newWhitelist` is the same as the current `whitelist` address.
  - ☐ Negative test
- Should revert if called by anyone other than the contract owner.
  - ☑ Negative test

5.20 Module: ScrollStandardERC20Factory.sol

Function: `deployL2Token(address _gateway, address _l1Token)`

Should be called by owner to deploy a new L2 token when a new one is added.

Inputs

- `_gateway`
  - **Control**: Fully controlled by the owner.
  - **Constraints**: None.
  - **Impact**: The gateway that the token is deployed for.
- `_l1Token`
  - **Control**: Fully controlled by the owner.
  - **Constraints**: None.
  - **Impact**: The L1 token that the token is deployed for.

Branches and code coverage (including function calls)

Intended branches

- Clone deterministically and return the address of the deployed token.
  ☑ Test coverage

Negative behavior

- Revert if the implementation address is zero.
  ☐ Negative test
- Revert if a token for that specific salt already exists. Currently not implemented.
  ☐ Negative test
- Revert if caller is not the owner.
  ☑ Negative test

5.21 Module: ScrollStandardERC20.sol

Function: `burn(address _from, uint256 _amount)`

Facilitates burning of tokens on the L2 side of the bridge, when the tokens are being moved from L2 to L1.

Inputs

- `_from`
  - **Control**: Controlled in code via gateway contract.
  - **Constraints**: None.
  - **Impact**: Balance of `_from` will be reduced by `_amount`.
- `_amount`
  - **Control**: Controlled in code via gateway contract.
  - **Constraints**: None.
  - **Impact**: Balance of `_from` will be reduced by `_amount`.

Branches and code coverage (including function calls)

Intended branches

- Should deplete the balance of `_from` by `_amount`.
  ☑ Test coverage
- Should only be used as part of the bridging process.
  ☑ Test coverage
- Assumes that the caller is the gateway.
  ☑ Test coverage

Negative behavior

- Should not be callable by anyone other than the gateway.
  ☑ Negative test

Function: `initialize(string _name, string _symbol, uint8 _decimals, address _gateway, address _counterpart)`

Initializes the token.

Inputs

- `_name`
  - **Control**: Full control.
  - **Constraints**: None.
  - **Impact**: The name of the token.
- `_symbol`
  - **Control**: Full control.
  - **Constraints**: None.
  - **Impact**: The symbol of the token.
- `_decimals`

  - **Control**: Full control.
  - **Constraints**: None.
  - **Impact**: The number of decimals of the token
- `_gateway`
  - **Control**: Full control.
  - **Constraints**: None.
  - **Impact**: The gateway that the token is deployed for.
- `_counterpart`
  - **Control**: Full control.
  - **Constraints**: None.
  - **Impact**: The counterpart address for this token.

Branches and code coverage (including function calls)

Intended branches

- Should call all underlying initializers; currently not done as it does not call ContextUpgradeable's initializer.
  ☐ Test coverage
- Set the name and symbol of the token.
  ☑ Test coverage
- Set the decimals of the token.
  ☑ Test coverage
- Set the gateway of the token.
  ☑ Test coverage
- Set the counterpart of the token.
  ☑ Test coverage

Negative behavior

- Should not be callable twice.
  ☑ Negative test

Function: `mint(address _to, uint256 _amount)`

Facilitates the creation of new tokens on the L2 side, when transferred from the L1 side.

Inputs

- `_to`
  - **Control**: Fully controlled by the gateway.

  - **Constraints**: None.
  - **Impact**: The address of the recipient of the minted tokens.
- `_amount`
  - **Control**: Fully controlled by the gateway.
  - **Constraints**: None.
  - **Impact**: Amount of tokens to mint.

Branches and code coverage (including function calls)

Intended branches

- Balance of `to` icreases by `_amount`.
  ☑ Test coverage

Negative behavior

- Should not be callable by anyone other than the gateway.
  ☑ Negative test
- Should not be callable in any other context than the bridging of tokens.
  ☐ Negative test

Function: `transferAndCall(address receiver, uint256 amount, byte[] data )`

Following the ERC677 standard, this function allows for the transfer of tokens to a contract that implements the `onTokenTransfer` callback.

Inputs

- `receiver`
  - **Control**: Fully controlled.
  - **Constraints**: None.
  - **Impact**: The address of the contract that will receive the tokens.
- `amount`
  - **Control**: Fully controlled.
  - **Constraints**: None.
  - **Impact**: Amount of tokens to transfer.
- `data`
  - **Control**: Fully controlled.
  - **Constraints**: None.
  - **Impact**: Calldata to pass to the `onTokenTransfer` callback.

Branches and code coverage (including function calls)

Intended branches

- Facilitate the `transfer` functionality from `msg.sender` to receiver.
  ☑ Test coverage
- Allow callback on to `receiver` after transfer occurs.
  ☑ Test coverage
- Assumes transfer is successful and that `receiver` is a contract. Currently does not check if transfer is successful.
  ☐ Test coverage
- Should increase the balance of `receiver` by `amount` and decrease the balance of `msg.sender` by `amount`.
  ☐ Test coverage

Negative behavior

- Should not be prone to reentrancy. Currently does not have a protection against reentrancy.
  ☐ Negative test

Function call analysis

- `receiver.onTokenTransfer(msg.sender, value, data)`
  - **What is controllable?** `receiver`, `data`, `amount`.
  - **If return value controllable, how is it used and how can it go wrong?** . n/a
  - **What happens if it reverts, reenters, or does other unusual control flow?** Should not do anything if it `receiver` is not a contract. Moreover, if `receiver` is a contract, it should trigger a callback. Assumes `receiver` inherits the `onTokenTransfer` callback.

6 Audit Results

At the time of our audit, the code was not deployed across different EVM-compatible chains.

During our audit, we discovered four findings. Of these, three were medium impact and the remaining finding was informational in nature. Scroll acknowledged all findings and implemented fixes.

6.1 Disclaimer

This assessment does not provide any warranties about finding all possible issues within its scope; in other words, the evaluation results do not guarantee the absence of any subsequent issues. Zellic, of course, also cannot make guarantees about any code added to the project after the audit version of our assessment. Furthermore, because a single assessment can never be considered comprehensive, we always recommend multiple independent assessments paired with a bug bounty program.

For each finding, Zellic provides a recommended solution. All code samples in these recommendations are intended to convey how an issue may be resolved (i.e., the idea), but they may not be tested or functional code.

Finally, the contents of this assessment report are for informational purposes only; do not construe any information in this report as legal, tax, investment, or financial advice. Nothing contained in this report constitutes a solicitation or endorsement of a project by Zellic.