Zellic

Starkgate

Smart Contract Security Assessment

October 19, 2023

Prepared for:

Haim Krasniker

StarkWare

Prepared by:

William Bowling, Ulrich Myhre, and Junyi Wang

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 StarkWare from October 3rd to October 17th, 2023. During this engagement, Zellic reviewed Starkgate's code for security vulnerabilities, design issues, and general weaknesses in security posture.

> Upon completing the audit, we were tasked with evaluating the modifications introduced in commit a4caa787. Our assessment revealed no security concerns arising from the changes made in this commit.

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:

- Do funds transfer correctly from L1 to L2?
- Are users able to cancel stuck transactions from L1 to L2?
- Do transfers from L2 to L1 work as designed?
- Is there any situation where funds are permanently stuck?
- Is a malicious actor able to transfer funds to themselves?

1.2 Non-goals and Limitations

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

- Underlying chain infrastructure

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

1.3 Results

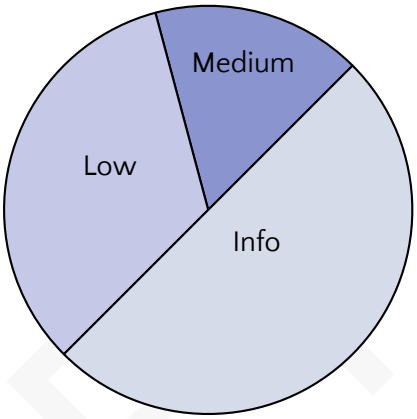
During our assessment on the scoped Starkgate contracts, we discovered six findings. No critical issues were found. One finding was of medium impact, two were of low impact, and the remaining findings were informational in nature.

Additionally, Zellic recorded its notes and observations from the assessment for

StarkWare'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	1
Low	2
Informational	3



2 Introduction

2.1 About Starkgate

Starkgate is the home for StarkNet's L1 bridges which enable you to fund your L2 wallet with ETH and ERC-20 tokens that reside on L1.

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 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 codebase 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.

Finally, Zellic provides a list of miscellaneous observations that do not have security impact or are not directly related to the scoped contracts itself. These observations — found in the Discussion (4) section of the document — may include suggestions for improving the codebase, or general recommendations, but do not necessarily convey that we suggest a code change.

2.3 Scope

The engagement involved a review of the following targets:

Starkgate Contracts

Repository	https://github.com/starknet-io/starkgate-contracts
Version	starkgate-contracts: `78b73afe5de30a0cd6d7e05bce8ca110be57d10a`
Programs	TokenBridge PermissionedERC20 StarknetTokenBridge StarkgateRegistry StarkgateManager StarknetEthBridge
Types	Solidity, Cairo
Platforms	EVM-compatible, Starknet

2.4 Project Overview

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

Contact Information

The following project manager was associated with the engagement:

Chad McDonald, Engagement Manager
chad@zellic.io

The following consultants were engaged to conduct the assessment:

William Bowling, Engineer
vakzz@zellic.io

Ulrich Myhre, Engineer
unblvr@zellic.io

Junyi Wang, Engineer
junyi@zellic.io

2.5 Project Timeline

The key dates of the engagement are detailed below.

October 5, 2023	Kick-off call
October 3, 2023	Start of primary review period
October 17, 2023	End of primary review period

3 Detailed Findings

3.1 ETH withdraw limit is non functional

- **Target**: token_bridge.cairo
- **Category**: Coding Mistakes
- **Severity**: Medium
- **Likelihood**: N/A
- **Impact**: Medium

Description

When withdrawing ETH from L2 to L1, on the destination chain there is a check shown below.

```
function withdraw(
    address token,
    uint256 amount,
    address recipient
) public {
    // [...]
    WithdrawalLimit.consumeWithdrawQuota(token, amount);
}
```

Following the call chain into `consumeWithdrawQuota`, eventually we arrive at the following function call:

```
function calculateIntradayAllowance(address token)
    internal view returns (uint256) {
    uint256 currentBalance = IERC20(token).balanceOf(address(this));
```

The `token` address used for ETH is `0` in these contracts.

Impact

This causes a revert and prevents withdraws when withdrawal limits are enabled.

Recommendations

Add a special case for ETH tokens.

Remediation

This issue has been acknowledged by StarkWare, and a fix was implemented in commit eedee830.

3.2 Token removal from registry

- **Target**: StarknetTokenBridge
- **Category**: Coding Mistakes
- **Likelihood**: Low
- **Severity**: Low
- **Impact**: Low

Description

The `checkDeploymentStatus` is used to update the status of pending tokens, either setting them to active or removing them from the registry.

```
function checkDeploymentStatus(address token)
    internal skipUnlessPending(token) {
    TokenSettings storage settings = tokenSettings()[token];
    bytes32 msgHash = settings.deploymentMsgHash;

    if (messagingContract().l1ToL2Messages(msgHash) == 0) {
        settings.tokenStatus = TokenStatus.Active;
    } else if (block.timestamp > settings.pendingDeploymentExpiration) {
        delete tokenSettings()[token];
        address registry = IStarkgateManager(manager()).getRegistry();
        IStarkgateRegistry(registry).selfRemove(token);
    }
}
```

The `checkDeploymentStatus` function is called from the `deposit` function when depositing tokens to the L1 bridge.

```
function deposit(
    address token,
    uint256 amount,
    uint256 l2Recipient
) external payable onlyServicingToken(token) {
    uint256 fee = acceptDeposit(token, amount);
    uint256[] memory noMessage = new uint256[](0);
    sendDepositMessage(token, amount, l2Recipient, noMessage,
    HANDLE_DEPOSIT_SELECTOR, fee);

    // Piggy-bag the deposit tx to check and update the status of token
    bridge deployment.
```

```
    checkDeploymentStatus(token);
}
```

The issue is that if a token deployment fails for some reason and `settings.pendingDeploymentExpiration` has passed, the only way to remove it from the registry is to call the `deposit` function with a token that will immediately be removed.

Impact

The only way to remove a token from the registry is to deposit a token that will then be immediately be removed from the registry, requiring the user to wait 5 days to reclaim the deposited tokens.

Recommendations

Either expose the `checkDeploymentStatus` function so that it can be called without depositing a token, or prevent the deposit from being made if the token is about to be removed from the registry.

Remediation

This issue has been acknowledged by StarkWare, and a fix was implemented in commit eedee830.

3.3 Implementation method clashes with proxy method

- **Target**: token_bridge.cairo
- **Category**: Coding Mistakes
- **Likelihood**: Low
- **Severity**: Low
- **Impact**: Low

Description

The token bridge has a method to remove a pending implementation from the mappings:

```
// Removes an existing implementation.
fn remove_implementation(ref self: ContractState,
    implementation_data: ImplementationData) {
    // The call is restricted to the upgrade governor.
    self.only_upgrade_governor();

    // Read implementation activation time.
    let impl_activation_time
    = self.get_impl_activation_time(:implementation_data);

    if (impl_activation_time.is_non_zero()) {
        self.set_impl_activation_time(:implementation_data,
    activation_time: 0);
        self.set_impl_expiration_time(:implementation_data,
    expiration_time: 0);
        self
            .emit(
                Event::ImplementationRemoved(ImplementationRemoved {
    implementation_data })
            );
    }
}
```

The issue is that some existing bridges are deployed using a proxy that has a method with the same name:

```
@external
func remove_implementation{syscall_ptr: felt*, pedersen_ptr:
    HashBuiltin*, range_check_ptr}(
    implementation_hash_: felt,
```

```
    eic_hash: felt,
    init_vector_len: felt,
    init_vector: felt*,
    final: felt,
) {
```

This means that the implementation method will never be called as the proxy method will always be used instead.

Impact

As the `calc_impl_key` methods for each contract are slightly different, trying to remove an implementation that was added using `add_new_implementation` will not work, as the proxy method will be called instead and return since it will not find a key.

Recommendations

Rename the implementation method to something that will not clash with the proxy method, similar to how `add_new_implementation` is being used instead of `add_implementation` in the proxy.

Remediation

This issue has been acknowledged by StarkWare.

StarkWare provided the following response:

> We consider the transition proxy -> no-proxy as a transient one, in which we assume the risk of not being able to remover an implementation.

3.4 Redundant overflow check

- **Target**: StarknetTokenBridge
- **Category**: Code Maturity
- **Likelihood**: N/A
- **Severity**: Informational
- **Impact**: Informational

Description

The following code is in `StarknetTokenBridge`.

```
// Virtual functions.
function acceptDeposit(address token, uint256 amount)
    internal virtual returns (uint256) {
    uint256 currentBalance = IERC20(token).balanceOf(address(this));
    require(currentBalance ≤ currentBalance + amount, "OVERFLOW");
    require(currentBalance + amount ≤ getMaxTotalBalance(token),
    "MAX_BALANCE_EXCEEDED");
```

The first require is not necessary, since `currentBalance + amount` reverts on overflow in Solidity 0.8 and above.

Impact

Gas is unnecessarily wasted.

Recommendations

Remove the first assert.

Remediation

This issue has been acknowledged by StarkWare, and a fix was implemented in commit eedee830.

3.5 Unused role check functions

- **Target**: token_bridge.cairo
- **Category**: Code Maturity
- **Likelihood**: N/A
- **Severity**: Informational
- **Impact**: Informational

Description

The token bridge has numerous roles that are setup in the `_initialize_roles` function:

```
self._set_role_admin(role: APP_GOVERNOR, admin_role: APP_ROLE_ADMIN);
self._set_role_admin(role: APP_ROLE_ADMIN, admin_role: GOVERNANCE_ADMIN);
self._set_role_admin(role: GOVERNANCE_ADMIN,
    admin_role: GOVERNANCE_ADMIN);
self._set_role_admin(role: OPERATOR, admin_role: APP_ROLE_ADMIN);
self._set_role_admin(role: TOKEN_ADMIN, admin_role: APP_ROLE_ADMIN);
self._set_role_admin(role: UPGRADE_GOVERNOR,
    admin_role: GOVERNANCE_ADMIN);
```

There are also matching sets of functions to assert that the current caller has the correct role. Currently, the `only_app_role_admin`, `only_governance_admin`, `only_operator`, and `only_token_admin` functions are never used.

Impact

While there is no security impact in having unused functions, having unused functions can make the codebase harder to understand and make maintenance more difficult.

Recommendations

The unused functions should be removed and potentially the roles should be simplified if they are not needed.

Remediation

This issue has been acknowledged by StarkWare.

StarkWare provided the following response:

> This is the full and generic initialization of our role component

3.6 Withdrawing from L2 to L1 with a zero recipient burns the tokens on L2

- **Target**: token_bridge.cairo
- **Category**: Code Maturity
- **Likelihood**: N/A
- **Severity**: Informational
- **Impact**: Informational

Description

When withdrawing tokens from L2 to L1, on the destination chain there is a check shown below.

```
function withdraw(
    address token,
    uint256 amount,
    address recipient
) public {
    // Make sure we don't accidentally burn funds.
    require(recipient ≠ address(0x0), "INVALID_RECIPIENT");
```

However, there is no check in the corresponding L2 function shown below.

```
// Initiates an l2-to-l1 token withdraw.
fn initiate_withdraw(
    ref self: ContractState, l1_recipient: EthAddress, token: EthAddress,
    amount: u256
) {
```

Impact

This causes tokens sent to the zero address from L2 to L1 to be burned anyway despite the check. The message is delivered, but cannot be consumed because of the revert. There exists no mechanism to cancel a message from L2 to L1 at the time of the finding.

Recommendations

Add a check that `l1_recipient` is nonzero on L2, preventing the burning of coins. Alternatively, remove the revert on L1 and allow the message to be consumed, burning the coins.

Remediation

This issue has been acknowledged by StarkWare, and a fix was implemented in commit eedee830.

4 Discussion

The purpose of this section is to document miscellaneous observations that we made during the assessment. These discussion notes are not necessarily security-related, and do not convey that we are suggesting a code change.

4.1 Missing test coverage

In our assessment of Starkgate test suite, we observed that while it provides adequate coverage for many aspects of the codebase, there are specific branches and codepaths that appear to be undertested or not covered at all.

WithdrawalLimit.sol: This newly added library is not covered by any test cases. It is a relatively simple library, but it is still important to have test coverage to ensure it functions as intended.

permissioned_erc20.cairo: This contract has good test cases for the permissioned minting and burning functionality, but is missing test cases for the other functions. While this contract is based on the OpenZeppelin code base, if the test are not being run against the modified version then any regressions will not be caught.

Negative testing: Incorporating negative testing can reveal vulnerabilities and improve the robustness of the codebase by ensuring it gracefully handles unexpected inputs and errors. In our experience, negative test cases catch a surprising amount of the more serious bugs. Some contracts and modules, like the Cairo and Solidity token bridges, are well-covered with test cases, but others are missing negative testing.

In our assessment, we find that enhancing test coverage for these specific areas would further bolster the reliability and resilience of the project. We recommend expanding the test suite to include test cases addressing the above points.

Remediation

This issue has been acknowledged by StarkWare, and a fix was implemented in commit eedee830.

5 Threat Model

This provides a full threat model description for various functions. As time permitted, we analyzed each function in the 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.

> Please note that our threat model was based on commit 78b73afe, which represents a specific snapshot of the codebase. Therefore, it's important to understand that the absence of certain tests in our report may not reflect the current state of the test suite.
>
> During the remediation phase, StarkWare took proactive steps to address the findings by adding test cases where applicable in commit eedee830. They also ensured that previously uncovered code branches were thoroughly tested with multiple tests. This demonstrates their dedication to enhancing the code quality and overall reliability of the system, which is commendable.

Function: `approve(ref self: TState, spender: core::starknet::contract_address::ContractAddress, amount: core::integer::u256)`

This function allows a certain amount of a user's tokens to be spent by another account.

Inputs

- `spender`
  - **Control**: Full control.
  - **Constraints**: Cannot be zero.
  - **Impact**: This account will be allowed to spend tokens on behalf of the caller.
- `amount`
  - **Control**: Full control.
  - **Constraints**: N/A.
  - **Impact**: This will be the amount of tokens that `spender` is allowed to use.

Branches and code coverage

Intended branches

- The allowance for the `spender` of the caller's tokens is set to `amount`.
  - ☐ Test coverage

Negative behavior

- The `spender` account is zero.
  - ☐ Negative test

Function: `decrease_allowance(ref self: TState, spender: core::starknet::contract_address::ContractAddress, subtracted_value: core::integer::u256)`

This function allows a user to decrease the allowance for a spender.

Inputs

- `spender`
  - **Control**: Full control.
  - **Constraints**: Cannot be zero.
  - **Impact**: This will be the account that has the allowance reduced.
- `subtracted_value`
  - **Control**: Full control.
  - **Constraints**: Must be less than or equal to the spender's current allowance.
  - **Impact**: This will be the amount that the allowance is reduced by.

Branches and code coverage

Intended branches

- The allowance of `spender` for the caller is reduced by `subtracted_value`.
  - ☐ Test coverage

Negative behavior

- The spender is zero.
  - ☐ Negative test
- The `subtracted_value` is more than the `spenders` current allowance.
  - ☐ Negative test

Function: `increase_allowance(ref self: TState, spender: core::starknet::contract_address::ContractAddress, added_value: core::integer::u256)`

This function allows a user to increase the allowance for a spender.

Inputs

- `spender`
  - **Control**: Full control.
  - **Constraints**: Cannot be zero.
  - **Impact**: This will be the account that has the allowance increased.
- `added_value`
  - **Control**: Full control.
  - **Constraints**: N/A.
  - **Impact**: This will be the amount that the allowance is increased by.

Branches and code coverage

Intended branches

- The allowance of `spender` for the caller is increased by `added_value`.
  - ☐ Test coverage

Function: `permissioned_burn(ref self: TState, account: core::starknet::contract_address::ContractAddress, amount: core::integer::u256)`

This function allows new tokens to be burned for an account. Only callable by the `permitted_minter`.

Inputs

- `account`
  - **Control**: Full control.
  - **Constraints**: Cannot be address zero.
  - **Impact**: This account will have `amount` tokens removed.
- `amount`
  - **Control**: Full control.
  - **Constraints**: Must be equal or less to the current balance of `account`.
  - **Impact**: This will be the amount of tokens removed from the account.

Branches and code coverage

Intended branches

- The `account` has `amount` tokens removed from their balance.
  ☑ Test coverage

Negative behavior

- The caller is not the permitted minter.
  ☑ Negative test
- The amount is greater than the balance of the account.
  ☑ Negative test

Function: `permissioned_mint(ref self: TState, account: core::starknet::contract_address::ContractAddress, amount: core::integer::u256)`

This function allows new tokens to be minted for an account. Only callable by the `permitted_minter`.

Inputs

- `account`
  - **Control**: Full control.
  - **Constraints**: N/A.
  - **Impact**: This will be the amount of tokens minted to the account.
- `amount`
  - **Control**: Full control.
  - **Constraints**: Cannot be the zero address.
  - **Impact**: This will be the account that receives the tokens.

Branches and code coverage

Intended branches

- The `account` has `amount` tokens added to their balance.
  ☑ Test coverage

Negative behavior

- The caller is not the permitted minter.
  ☑ Negative test
- The amount plus the current account balance is larger than the maximum value of a `u256`.
  ☑ Negative test

Function: `transfer(ref self: TState, recipient: core::starknet::contract_address::ContractAddress, amount: core::integer::u256)`

This function transfers `amount` tokens from the caller to `recipient`.

Inputs

- `recipient`
  - **Control**: Full control.
  - **Constraints**: Cannot be the zero address.
  - **Impact**: This account will receive `amount` tokens.
- `amount`
  - **Control**: Full control.
  - **Constraints**: Must be less than or equal to the caller's balance.
  - **Impact**: The `recipient` will receive this amount of tokens, and the caller's balance will be reduced.

Branches and code coverage

Intended branches

- The `recipient` receives `amount` tokens, and the caller's balance is reduced by `amount`.
  - ☐ Test coverage

Negative behavior

- The caller does not have enough balance.
  - ☐ Negative test

Function: `transferFrom(ref self: TState, sender: core::starknet::contract_address::ContractAddress, recipient: core::starknet::contract_address::ContractAddress, amount: core::integer::u256)`

This function allows a user to transfer tokens to an account on behalf of another account.

Inputs

- `sender`
  - **Control**: Full control.
  - **Constraints**: The caller must have been approved for `amount` by the sender.
  - **Impact**: This account will be where the tokens are transferred from.

- `recipient`
  - **Control**: Full control.
  - **Constraints**: Cannot be the zero address.
  - **Impact**: This address will receive the tokens.
- `amount`
  - **Control**: Full control.
  - **Constraints**: Must be less than or equal to the senders balance.
  - **Impact**: This amount of tokens will be transferred from `sender` to `recipient` and the caller's allowance for `sender` will be reduced.

Branches and code coverage

Intended branches

- There are `amount` tokens transferred from `sender` to `recipient`, and the caller's allowance for `sender` is reduced.
  - ☐ Test coverage

Negative behavior

- The caller does not have enough allowance for `sender`.
  - ☐ Negative test
- The `sender` does not have enough tokens.
  - ☐ Negative test

Function: `transfer_from(ref self: TState, sender: core::starknet::contract_address::ContractAddress, recipient: core::starknet::contract_address::ContractAddress, amount: core::integer::u256)`

This function allows a user to transfer tokens to an account on behalf of another account.

Inputs

- `sender`
  - **Control**: Full control.
  - **Constraints**: The caller must have been approved for `amount` by the sender.
  - **Impact**: This account will be where the tokens are transferred from.
- `recipient`
  - **Control**: Full control.
  - **Constraints**: Cannot be the zero address.
  - **Impact**: This address will receive the tokens.

- `amount`
  - **Control**: Full control.
  - **Constraints**: Must be less than or equal to the senders balance.
  - **Impact**: This amount of tokens will be transferred from `sender` to `recipient` and the caller's allowance for `sender` will be reduced.

Branches and code coverage

Intended branches

- There are `amount` tokens transferred from `sender` to `recipient`, and the caller's allowance for `sender` is reduced.
  - ☐ Test coverage

Negative behavior

- The caller does not have enough allowance for `sender`.
  - ☐ Negative test
- The `sender` does not have enough tokens.
  - ☐ Negative test

5.1 Module: StarkgateManager.sol

Function: `addExistingBridge(address token, address bridge_)`

Add an existing token/bridge pair to be managed by the contract.

Inputs

- `token`
  - **Constraints**: Token/bridge pair not already added to registry.
  - **Impact**: The token that will be added.
- `bridge_`
  - **Constraints**: Token/bridge pair not already added to registry.
  - **Impact**: The bridge that will be added.

Branches and code coverage

Intended branches

- The tokens to bridge mapping in the registry is updated.
  - ☐ Test coverage
- The withdrawal bridges mapping in the registry is updated.

☑ Test coverage

Negative behavior

- Reverts when called by non-admin.
  ☐ Negative test
- Reverts when token already enrolled or deactivated.
  ☐ Negative test

Function call analysis

- `addExistingBridge → IStarkgateRegistry(registry()).enlistToken(token, bridge_)`
  - **What is controllable?** `token`.
  - **If the return value is controllable, how is it used and how can it go wrong?** Unused.
  - **What happens if it reverts, reenters or does other unusual control flow?** `registry` is set by admin to a trusted contract. In the current implementation, it can not reenter.

Function: `blockToken(address token)`

Prevents a token from ever being registered.

Inputs

- `token`
  - **Constraints**: Token must not currently be registered to a bridge.
  - **Impact**: This token will be blocked.

Branches and code coverage

Intended branches

- Token can no longer be registered after blocking.
  ☐ Test coverage

Negative behavior

- Reverts when called by non-admin.
  ☐ Negative test
- Reverts if token is already in service.
  ☐ Negative test

Function call analysis

- `blockToken(token)` → `registryContract.blockToken(token)`
  - **What is controllable?** `token`
  - **If the return value is controllable, how is it used and how can it go wrong?** Unused.
  - **What happens if it reverts, reenters or does other unusual control flow?** One-line function; no unusual control flow possible.

Function: `deactivateToken(address token)`

Marks a token which is currently "enrolled" as non-operative.

Inputs

- `token`
  - **Constraints**: The bridge for the token exists in the registry
  - **Impact**: The token that will be deactivated

Branches and code coverage

Intended branches

- Token no longer allows deposits after deactivation.
  ☐ Test coverage

Negative behavior

- Reverts when called by non-admin.
  ☑ Negative test
- Reverts if token is not already registered.
  ☐ Negative test
- Reverts if token redeployed.
  ☑ Negative test

Function call analysis

- `deactivateToken(token)` → `registryContract.deactivateToken(token)`
  - **What is controllable?** `token`. The caller verifies that it is registered to a bridge.
  - **If the return value is controllable, how is it used and how can it go wrong?** Unused.
  - **What happens if it reverts, reenters or does other unusual control flow?** One-line function; no unusual control flow possible.

- `deactivateToken(token)` → `IStarkgateBridge(bridge()).deactivate(token)`
  - **What is controllable?** `token`. The caller verifies that it is registered to a bridge.
  - **If the return value is controllable, how is it used and how can it go wrong?** Unused.
  - **What happens if it reverts, reenters or does other unusual control flow?** The bridge is set by `admin` to a contract which has no external calls in the `deactivate` function.

Function: `enrollTokenBridge(address token)`

Enrolls a token into the bridge managed by the current contract.

Inputs

- `token`
  - **Constraints**: Token is not blocked.
  - **Impact**: The token is enrolled into the bridge.

Branches and code coverage

Intended branches

- The bridge goes into pending state for the token.
  ☑ Test coverage
- The tokens to bridge mapping in the registry is updated.
  ☑ Test coverage
- The withdrawal bridges mapping in the registry is updated.
  ☑ Test coverage

Negative behavior

- Reverts if the token is blocked or deactivated.
  ☑ Negative test
- Reverts when called by non-admin.
  ☐ Negative test
- Reverts when token already enrolled.
  ☑ Negative test

Function call analysis

- `enrollTokenBridge` → `registryContract.enlistToken(token, bridge())`
  - **What is controllable?** `token`

- **If the return value is controllable, how is it used and how can it go wrong?** Unused.
- **What happens if it reverts, reenters or does other unusual control flow?** `registry` is set by admin to a trusted contract. In the current implementation, it can not reenter.

5.2 Module: StarkgateRegistry.sol

Function: `blockToken(address token)`

See `blockToken` in `StarkgateManager`.

Branches and code coverage

Negative behavior

- Reverts when called by non-manager.
  - ☐ Negative test

Function: `deactivateToken(address token)`

See `deactivateToken` in `StarknetManager`.

Branches and code coverage

Negative behavior

- Reverts when called by non-manager.
  - ☐ Negative test

Function: `enlistToken(address tokenAddress, address bridge)`

See `enlistToken` in `StarknetManager`

Branches and code coverage

Negative behavior

- Reverts when called by non-manager.
  - ☑ Negative test

Function: `selfRemove(address token)`

Used by the bridge itself to remove a token it is servicing from the registry.

Inputs

- `token`
  - **Constraints**: None.
  - **Impact**: The token's bridge mapping is cleared.

Branches and code coverage

Intended branches

- The token's bridge mapping is cleared.
  - ☐ Test coverage

Negative behavior

- Caller must be the bridge of the token.
  - ☐ Negative test
- The bridge must claim that the token is no longer in service.
  - ☐ Negative test

5.3 Module: StarknetTokenBridge.sol

Function: `deactivate(address token)`

See `deactivateToken` in `StarknetManager`.

Branches and code coverage

Negative behavior

- Reverts when called by non-manager.
  - ☐ Negative test

Function: `depositCancelRequest(address token, uint256 amount, uint256 l2Recipient, uint256 nonce)`

Starts deposit cancellation, given the original parameters.

Inputs

- `token`
  - **Constraints**: None, but must match nonce.
  - **Impact**: Used to verify nonce.
- `amount`

  - **Constraints**: None, but must match nonce.
  - **Impact**: Used to verify nonce.
- `l2Recipient`
  - **Constraints**: None, but must match nonce.
  - **Impact**: Used to verify nonce.
- `nonce`
  - **Constraints**: Must be the nonce of a valid message which was sent by the caller.
  - **Impact**: The nonce of the deposit request to be cancelled.

Branches and code coverage

Intended branches

- Cancellation request is sent.
  ☑ Test coverage

Negative behavior

- Caller is the depositor.
  ☑ Negative test
- Parameters given match the nonce.
  ☑ Negative test

Function call analysis

- `depositCancelRequest` → `messagingContract().startL1ToL2MessageCancellation`
  - **What is controllable?** message payload, `nonce`
  - **If the return value is controllable, how is it used and how can it go wrong?** Unused.
  - **What happens if it reverts, reenters or does other unusual control flow?** Parent function has no other side effects. Reentrancy is not a problem.

Function: `depositReclaim(address token, uint256 amount, uint256 l2Recipient, uint256 nonce)`

Reclaims a deposit after a cancellation request is submitted and the timer expires.

Inputs

- `token`
  - **Constraints**: None, but must match nonce.

  - **Impact**: Used to verify nonce.
- `amount`
  - **Constraints**: None, but must match nonce.
  - **Impact**: Used to verify nonce.
- `l2Recipient`
  - **Constraints**: None, but must match nonce.
  - **Impact**: Used to verify nonce.
- `nonce`
  - **Constraints**: Must be the nonce of a valid message which was sent by the caller.
  - **Impact**: The nonce of the deposit request to be cancelled.

Branches and code coverage

Intended branches

- Returns the funds to the sender.
  ☑ Test coverage

Negative behavior

- Caller is the depositor.
  ☑ Negative test
- Message matches the nonce.
  ☑ Negative test

Function call analysis

- `depositReclaim` → `messagingContract().cancelL1ToL2Message`
  - **What is controllable?** message payload, `nonce`
  - **If the return value is controllable, how is it used and how can it go wrong?** Unused.
  - **What happens if it reverts, reenters or does other unusual control flow?** The message must successfully cancel before refunds are given. Reentrancy is not a problem.

Function: `depositWithMessageCancelRequest(address token, uint256 amount, uint256 l2Recipient, uint256[] message, uint256 nonce)`

Starts deposit cancellation for deposits with message, given the original parameters.

Inputs

- `token`
  - **Constraints**: None, but must match nonce.
  - **Impact**: Used to verify nonce.
- `amount`
  - **Constraints**: None, but must match nonce.
  - **Impact**: Used to verify nonce.
- `l2Recipient`
  - **Constraints**: None, but must match nonce.
  - **Impact**: Used to verify nonce.
- `message`
  - **Constraints**: None, but must match nonce.
  - **Impact**: Used to verify nonce.
- `nonce`
  - **Constraints**: Must be the nonce of a valid message which was sent by the caller.
  - **Impact**: The nonce of the deposit request to be cancelled.

Branches and code coverage

Intended branches

- Cancellation request is sent.
  ☑ Test coverage

Negative behavior

- Caller is the depositor.
  ☑ Negative test
- Parameters given match the nonce.
  ☑ Negative test

Function call analysis

- `depositCancelRequest` → `messagingContract().startL1ToL2MessageCancellation`
  - **What is controllable?** message payload, `nonce`
  - **If the return value is controllable, how is it used and how can it go wrong?** Unused.
  - **What happens if it reverts, reenters or does other unusual control flow?** Parent function has no other side effects. Reentrancy is not a problem.

Function: `depositWithMessageReclaim(address token, uint256 amount, uint256 l2Recipient, uint256[] message, uint256 nonce)`

Reclaims a deposit with message after a cancellation request is submitted and the timer expires.

Inputs

- `token`
  - **Constraints**: None, but must match nonce.
  - **Impact**: Used to verify nonce.
- `amount`
  - **Constraints**: None, but must match nonce.
  - **Impact**: Used to verify nonce.
- `l2Recipient`
  - **Constraints**: None, but must match nonce.
  - **Impact**: Used to verify nonce.
- `message`
  - **Constraints**: None, but must match nonce.
  - **Impact**: Used to verify nonce.
- `nonce`
  - **Constraints**: Must be the nonce of a valid message which was sent by the caller.
  - **Impact**: The nonce of the deposit request to be cancelled.

Branches and code coverage

Intended branches

- Returns the funds to the sender.
  ☑ Test coverage

Negative behavior

- Caller is the depositor.
  ☑ Negative test
- Message matches the nonce.
  ☑ Negative test

Function call analysis

- `depositReclaim` → `messagingContract().cancelL1ToL2Message`
  - **What is controllable?** message payload, `nonce`

  - **If the return value is controllable, how is it used and how can it go wrong?** Unused.
  - **What happens if it reverts, reenters or does other unusual control flow?** The message must successfully cancel before refunds are given. Reentrancy is not a problem.

Function: `depositWithMessage(address token, uint256 amount, uint256 l2Recipient, uint256[] message)`

Deposits the amount into the bridge, and sends the amount on L2 to the `l2Recipient` with a message.

Inputs

- `token`
  - **Constraints**: Only tokens that are in service allowed.
  - **Impact**: The token that is deposited.
- `amount`
  - **Constraints**: Must be able to transfer this amount of the token into the contract.
  - **Impact**: This is the amount of the token that will be deposited.
- `l2Recipient`
  - **Constraints**: The address must be contained in a field element.
  - **Impact**: The receiver on L2.
- `message`
  - **Constraints**: Must all be field elements.
  - **Impact**: This message is sent along with the funds to L2.

Branches and code coverage

Intended branches

- Sends the deposit message to L2.
  ☑ Test coverage
- If token is pending and successfully deployed on L2, it becomes active.
  ☑ Test coverage
- If token is not deployed and the deployment timer has expired, it is removed from the registry.
  ☑ Test coverage

Negative behavior

- Reverts when called with an unserviced token.

☑ Negative test

- Reverts when the user does not provide sufficient funds.
  ☑ Negative test

Function call analysis

- `depositWithMessage` → `Transfers.transferIn(token, msg.sender, amount)`
  - **What is controllable?** `token`, `amount`
  - **If the return value is controllable, how is it used and how can it go wrong?** Unused.
  - **What happens if it reverts, reenters or does other unusual control flow?** Does not reenter.

Function: `deposit(address token, uint256 amount, uint256 l2Recipient)`

Deposits the amount into the bridge, and sends the amount on L2 to the `l2Recipient`.

Inputs

- `token`
  - **Constraints**: Only tokens that are in service allowed.
  - **Impact**: The token that is deposited.
- `amount`
  - **Constraints**: Must be able to transfer this amount of the token into the contract.
  - **Impact**: This is the amount of the token that will be deposited.
- `l2Recipient`
  - **Constraints**: The address must be contained in a field element.
  - **Impact**: The receiver on L2.

Branches and code coverage

Intended branches

- Sends the deposit message to L2.
  ☑ Test coverage
- If token is pending and successfully deployed on L2, it becomes active.
  ☑ Test coverage
- If token is not deployed and the deployment timer has expired, it is removed from the registry.
  ☐ Test coverage

Negative behavior

- Reverts when called with an unserviced token.
  ☑ Negative test
- Reverts when the user does not provide sufficient funds.
  ☑ Negative test

Function call analysis

- `deposit` → `Transfers.transferIn(token, msg.sender, amount)`
  - **What is controllable?** `token`, `amount`
  - **If the return value is controllable, how is it used and how can it go wrong?** Unused.
  - **What happens if it reverts, reenters or does other unusual control flow?** Does not reenter.

Function: `enrollToken(address token)`

See `enrollTokenBridge` in `StarknetManager`.

Branches and code coverage

Negative behavior

- Reverts when called by non-manager.
  ☐ Negative test

Function: `setL2TokenBridge(uint256 l2TokenBridge_)`

Sets the L2 counterpart of this bridge.

Inputs

- `l2TokenBridge_`
  - **Constraints**: Must be a valid L2 address.
  - **Impact**: The new L2 token bridge.

Branches and code coverage

Intended branches

- Sets the L2 token bridge.
  ☑ Test coverage

Negative behavior

- Caller is a governance admin.
  ☑ Negative test
- L1 bridge is initialized.
  ☑ Negative test

Function: `setMaxTotalBalance(address token, uint256 maxTotalBalance_)`

Set the maximum allowed balance of the bridge.

Inputs

- `token`
  - **Constraints**: None.
  - **Impact**: The token for which the max total balance is set.
- `maxTotalBalance_`
  - **Constraints**: Nonzero.
  - **Impact**: The new maximum balance.

Branches and code coverage

Intended branches

- Sets the maximum total balance to the requested value.
  ☑ Test coverage

Negative behavior

- Caller is a governance admin.
  ☑ Negative test
- Maximum total balance is nonzero.
  ☑ Negative test

Function: `setWithdrawalLimitApplied(address token, bool withdrawalLimitApplied_)`

The description

Inputs

- `token`
  - **Constraints**: None.
  - **Impact**: The token that is modified.

- `withdrawalLimitApplied_`
  - **Constraints**: None.
  - **Impact**: The new boolean value of whether the withdraw limit applies.

Branches and code coverage

Intended branches

- The withdraw limit boolean is set to the requested value.
  - ☐ Test coverage

Negative behavior

- Caller is a governance admin.
  - ☐ Negative test

Function: `withdraw(address token, uint256 amount)`

Withdraws tokens to the caller. Requires an L2 message with the same parameters to be already received.

See `withdraw(address token, uint256 amount, address recipient)`.

Function: `withdraw(address token, uint256 amount, address recipient)`

Withdraws tokens to the specified address. Requires an L2 message with the same parameters to be already received.

Inputs

- `token`
  - **Constraints**: None, but must match L2 message.
  - **Impact**: The token that is withdrawn.
- `amount`
  - **Constraints**: None, but must match L2 message.
  - **Impact**: The amount to withdraw.
- `recipient`
  - **Constraints**: None, but must match L2 message.
  - **Impact**: The receiving L1 address.

Branches and code coverage

Intended branches

- The recipient receives the funds in the specified amount.
  ☑ Test coverage

Negative behavior

- Reverts if no such L2 message exists.
  ☑ Negative test
- Reverts if the daily withdrawal limit has been reached
  ☐ Negative test

Function call analysis

- `withdraw → messagingContract().consumeMessageFromL2(l2TokenBridge(), payload)`
  - **What is controllable?** `payload`
  - **If the return value is controllable, how is it used and how can it go wrong?** Unused.
  - **What happens if it reverts, reenters or does other unusual control flow?** The messaging contract is a trusted standard library contract.
- `withdraw → Transfers.transferOut(token, recipient, amount)`
  - **What is controllable?** `token`, `recipient`, `amount`
  - **If the return value is controllable, how is it used and how can it go wrong?** Unused.
  - **What happens if it reverts, reenters or does other unusual control flow?** Does not reenter.
- `withdraw → WithdrawalLimit.consumeWithdrawQuota(token, amount)`
  - **What is controllable?** `token`, `amount`
  - **If the return value is controllable, how is it used and how can it go wrong?** Unused.
  - **What happens if it reverts, reenters or does other unusual control flow?** Does not reenter.

Function: `add_new_implementation(ref self: TState, implementation_data: src::replaceability_interface::ImplementationData)`

This function allows a new implementation for the contract to be added, which can be upgraded to once the `upgrade_delay` time has passed. The upgrade can only be performed within two weeks before it expires. Only callable by an account with the `UPGRADE_GOVERNOR`.

Inputs

- `implementation_data`
  - **Control**: Full control.
  - **Constraints**: N/A.
  - **Impact**: This holds the new class hash that the contract can be upgraded to, the eic (external initializer contract) data, as well as a flag to finalize the contract.

Branches and code coverage

Intended branches

- The new implementation is added to the `impl_activation_time` mapping with the time that it can be activated at.
  ☑ Test coverage

Negative behavior

- The caller does not have the `UPGRADE_GOVERNOR` role.
  ☑ Negative test

Function: `apply_withdrawal_limit(ref self: TState, token: core::starknet::eth_address::EthAddress, applied_state: core::bool)`

This function enables or disables a withdrawal limit for a bridged token, emitting either the `LimitWithdrawalOn` or `LimitWithdrawalOff` event. Only callable by an app governor.

Inputs

- `token`
  - **Control**: Full control.
  - **Constraints**: Must be an enrolled token.
  - **Impact**: This token will have the withdrawal limit enable or disabled.
- `applied_state`
  - **Control**: Full control.
  - **Constraints**: N/A.
  - **Impact**: This controls whether the withdrawal limit is enable or disabled..

Branches and code coverage

Intended branches

- The withdrawal limit for a token is enabled and the `LimitWithdrawalOn` event is emitted.
  ☑ Test coverage
- The withdrawal limit for a token is disabled and the `LimitWithdrawalOff` event is emitted.
  ☑ Test coverage

Negative behavior

- The caller is not an app governor.
  ☑ Negative test
- The token is not enrolled.
  ☑ Negative test

Function: `handle_deposit(ref self: ContractState, from_address: felt252, l2_recipient: ContractAddress, token: EthAddress, amount: u256)`

This L1 handler function allows the L1 bridge to mint the corresponding L2 token when a deposit is made to the L1 token bridge. Only callable by the L1 bridge.

Inputs

- `from_address`
  - **Control**: Will be the L1 address that sent the message.
  - **Constraints**: Must be the current `l1_bridge_address`.
  - **Impact**: Used for authorizing the cross-chain call.
- `l2_recipient`
  - **Control**: Full control.
  - **Constraints**: Cannot be the zero address.
  - **Impact**: Will be minted `amount` of `token`.
- `token`
  - **Control**: Full control.
  - **Constraints**: Must be enrolled in the bridge.
  - **Impact**: This token will be minted.
- `amount`
  - **Control**: Full control.
  - **Constraints**: N/A.
  - **Impact**: This amount of tokens will be minted to the `l2_recipient`.

Branches and code coverage

Intended branches

- The `l2_recipient` is minted `amount` of `token`.
  ☑ Test coverage

Negative behavior

- The caller is not the L1 bridge.
  ☑ Negative test
- The token is not enrolled in the bridge.
  ☐ Negative test

Function call analysis

- `handle_deposit_common` → `l2_token_address.permissioned_mint(l2_recipient, amount)`
  - **What is controllable?** `l2_recipient` and `amount` are fully controllable by the L1 bridge.
  - **If the return value is 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 the call reverts then the transaction will be aborted.

Function: `handle_deposit_with_message(ref self: ContractState, from_address: felt252, l2_recipient: ContractAddress, token: EthAddress, amount: u256, depositor: EthAddress, message: Span<felt252>)`

This L1 handler function allows the L1 bridge to mint the corresponding L2 token when a deposit is made to the L1 token bridge, then call the `on_receive` function of the `l2_recipient`. Only callable by the L1 bridge.

Inputs

- `from_address`
  - **Control**: Will be the L1 address that sent the message.
  - **Constraints**: Must be the current `l1_bridge_address`.
  - **Impact**: Used for authorizing the cross-chain call.
- `l2_recipient`
  - **Control**: Full control.
  - **Constraints**: Cannot be the zero address.
  - **Impact**: Will be minted `amount` of `token`.
- `token`
  - **Control**: Full control.
  - **Constraints**: Must be enrolled in the bridge.

  - **Impact**: This token will be minted.
- `amount`
  - **Control**: Full control.
  - **Constraints**: N/A.
  - **Impact**: This amount of tokens will be minted to the `l2_recipient`.
- `depositor`
  - **Control**: Full control.
  - **Constraints**: The L1 bridge will set this to the caller of the `depositWithMessage` function.
  - **Impact**: Will be sent in the call to 'l2_recipient.on_receive().
- `message`
  - **Control**: Full control.
  - **Constraints**: N/A.
  - **Impact**: Will be sent in the call to `l2_recipient.on_receive()`.

Branches and code coverage

Intended branches

- The `l2_recipient` is minted `amount` of `token` and then has its `on_receive` function called.
  ☑ Test coverage

Negative behavior

- The caller is not the L1 bridge.
  ☑ Negative test
- The token is not enrolled in the bridge.
  ☐ Negative test
- The call to `l2_recipient.on_receive` fails
  ☑ Negative test

Function call analysis

- `handle_deposit_with_message` → `l2_recipient.on_receive(l2_token_address, amount, depositor, message)`
  - **What is controllable?** `l2_token_address`, `amount`, `depositor`, and `message` are fully controllable by the L1 bridge.
  - **If the return value is controllable, how is it used and how can it go wrong?** The return value is used to check if the call succeeded.
  - **What happens if it reverts, reenters or does other unusual control flow?** If the call reverts then the transaction will be aborted.

Function: `handle_token_enrollment(ref self: ContractState, from_address: felt252, l1_token_address: EthAddress, name: felt252, symbol: felt252, decimals: u8)`

This L1 handler function allows the L1 token bridge to enroll a new token in the L2 bridge, creating the corresponding L2 token. Only callable by the L1 bridge.

Inputs

- `from_address`
  - **Control**: This will be the L1 address that sent the message.
  - **Constraints**: This must match the current `l1_bridge_address`.
  - **Impact**: Used for authorizing the cross-chain call.
- `l1_token_address`
  - **Control**: Full control.
  - **Constraints**: Must not already be enrolled in the bridge.
  - **Impact**: This L1 token will have a corresponding L2 token created and enrolled in the bridge.
- `name`
  - **Control**: Full control.
  - **Constraints**: N/A.
  - **Impact**: Will be the name of the L2 token.
- `symbol`
  - **Control**: Full control.
  - **Constraints**: N/A.
  - **Impact**: Will be the symbol of the L2 token.
- `decimals`
  - **Control**: Full control.
  - **Constraints**: N/A.
  - **Impact**: Will be the decimals of the L2 token.

Branches and code coverage

Intended branches

- The L1 token is not already enrolled, the L2 token is created, and the `DeployHandled` event is emitted.
  ☑ Test coverage

Negative behavior

- The message sender is not the L1 bridge.

☑ Negative test

- The token is already enrolled.
  ☑ Negative test

Function call analysis

- `handle_token_enrollment` → `deploy_syscall(class_hash, 0, calldata.span(), false)`
  - **What is controllable?** The `calldata` is fully controllable by the L1 bridge.
  - **If the return value is controllable, how is it used and how can it go wrong?** The return value is added to the mapping of L1 tokens to L2 tokens.
  - **What happens if it reverts, reenters or does other unusual control flow?** If the call fails then the transaction will be aborted.

Function: `initiate_withdraw(ref self: TState, l1_recipient: core::starknet::eth_address::EthAddress, token: core::starknet::eth_address::EthAddress, amount: core::integer::u256)`

This function allows a user to burn their tokens on L2 and withdraw then on the L1 token bridge once the message has propagated.

Inputs

- `l1_recipient`
  - **Control**: Full control.
  - **Constraints**: N/A.
  - **Impact**: This address will be sent `amount` of the corresponding L1 `token`.
- `token`
  - **Control**: Full control.
  - **Constraints**: The token must have been enrolled in the bridge.
  - **Impact**: This will be the token that is burnt and withdrawn on L1.
- `amount`
  - **Control**: Full control.
  - **Constraints**: Cannot be zero and the caller's balance must be greater than this amount. If the token has withdrawal limits enabled, this amount cannot make the daily withdrawal exceed 10% of the total supply.
  - **Impact**: This will be the amount of `token` that is burnt and withdrawn on L1.

Branches and code coverage

Intended branches

- The caller has `amount` tokens burnt and a message is sent to the L1 bridge with the appropriate data.
  - ☑ Test coverage

Negative behavior

- The caller does not have enough balance.
  - ☑ Negative test
- The token is not enrolled in the bridge.
  - ☐ Negative test
- The amount is zero.
  - ☑ Negative test

Function call analysis

- `initiate_withdraw` → `send_message_to_l1_syscall(l1_bridge_address, message_payload)`
  - **What is controllable?** The message payload will contain the `l1_recipient`, `token`, and `amount`.
  - **If the return value is controllable, how is it used and how can it go wrong?** The return value is used to ensure the call succeeded.
  - **What happens if it reverts, reenters or does other unusual control flow?** If the call reverts or returns a failure, the transaction will be aborted.

Function: `register_app_governor(ref self: TState, account: core::starknet::contract_address::ContractAddress)`

This function allows a user with the `APP_ROLE_ADMIN` role to grant the `APP_GOVERNOR` role to another account and will emit a `AppGovernorAdded` event.

Inputs

- `account`
  - **Control**: N/A.
  - **Constraints**: Cannot be the zero address.
  - **Impact**: This account will be granted the `APP_GOVERNOR` role.

Branches and code coverage

Intended branches

- The `account` is granted the `APP_GOVERNOR` role and a `AppGovernorAdded` event is emitted.
  ☑ Test coverage

Negative behavior

- The caller does not have the `APP_ROLE_ADMIN` role.
  ☐ Negative test
- The `account` is the zero address.
  ☐ Negative test

Function: `register_app_role_admin(ref self: TState, account: core::starknet::contract_address::ContractAddress)`

This function allows a user with the `GOVERNANCE_ADMIN` role to grant the `APP_ROLE_ADMIN` role to another account and will emit a `AppRoleAdminAdded` event.

Inputs

- `account`
  - **Control**: N/A.
  - **Constraints**: Cannot be the zero address.
  - **Impact**: This account will be granted the `APP_ROLE_ADMIN` role.

Branches and code coverage

Intended branches

- The `account` is granted the `APP_ROLE_ADMIN` role and a `AppRoleAdminAdded` event is emitted.
  ☑ Test coverage

Negative behavior

- The caller does not have the `GOVERNANCE_ADMIN` role.
  ☐ Negative test
- The `account` is the zero address.
  ☐ Negative test

Function: `register_governance_admin(ref self: TState, account: core::starknet::contract_address::ContractAddress)`

This function allows a user with the `GOVERNANCE_ADMIN` role to grant the `GOVERNANCE_ADMIN` role to another account and will emit a `GovernanceAdminAdded` event.

Inputs

- `account`
  - **Control**: N/A.
  - **Constraints**: Cannot be the zero address.
  - **Impact**: This account will be granted the `GOVERNANCE_ADMIN` role.

Branches and code coverage

Intended branches

- The `account` is granted the `GOVERNANCE_ADMIN` role and a `GovernanceAdminAdded` event is emitted.
  ☑ Test coverage

Negative behavior

- The caller does not have the `GOVERNANCE_ADMIN` role.
  ☐ Negative test
- The `account` is the zero address.
  ☐ Negative test

Function: `register_operator(ref self: TState, account: core::starknet::contract_address::ContractAddress)`

This function allows a user with the `APP_ROLE_ADMIN` role to grant the `OPERATOR` role to another account and will emit a `OperatorAdded` event.

Inputs

- `account`
  - **Control**: N/A.
  - **Constraints**: Cannot be the zero address.
  - **Impact**: This account will be granted the `OPERATOR` role.

Branches and code coverage

Intended branches

- The `account` is granted the `OPERATOR` role and a `OperatorAdded` event is emitted.
  ☑ Test coverage

Negative behavior

- The caller does not have the `APP_ROLE_ADMIN` role.

☐ Negative test

- The `account` is the zero address.
  ☐ Negative test

Function: `register_token_admin(ref self: TState, account: core::starknet::contract_address::ContractAddress)`

This function allows a user with the `APP_ROLE_ADMIN` role to grant the `TOKEN_ADMIN` role to another account and will emit a `TokenAdminAdded` event.

Inputs

- `account`
  - **Control**: N/A.
  - **Constraints**: Cannot be the zero address.
  - **Impact**: This account will be granted the `TOKEN_ADMIN` role.

Branches and code coverage

Intended branches

- The `account` is granted the `TOKEN_ADMIN` role and a `TokenAdminAdded` event is emitted.
  ☑ Test coverage

Negative behavior

- The caller does not have the `APP_ROLE_ADMIN` role.
  ☐ Negative test
- The `account` is the zero address.
  ☐ Negative test

Function: `register_upgrade_governor(ref self: TState, account: core::starknet::contract_address::ContractAddress)`

This function allows a user with the `GOVERNANCE_ADMIN` role to grant the `UPGRADE_GOVERNOR` role to another account and will emit a `UpgradeGovernorAdded` event.

Inputs

- `account`
  - **Control**: N/A.
  - **Constraints**: Cannot be the zero address.
  - **Impact**: This account will be granted the `UPGRADE_GOVERNOR` role.

Branches and code coverage

Intended branches

- The `account` is granted the `UPGRADE_GOVERNOR` role and a `UpgradeGovernorAdded` event is emitted.
  ☑ Test coverage

Negative behavior

- The caller does not have the `GOVERNANCE_ADMIN` role.
  ☐ Negative test
- The `account` is the zero address.
  ☐ Negative test

Function: `remove_app_governor(ref self: TState, account: core::starknet::contract_address::ContractAddress)`

This function allows a user with the `APP_ROLE_ADMIN` role to remove the `APP_GOVERNOR` role from another account and will emit an `AppGovernorRemoved` event.

Inputs

- `account`
  - **Control**: Full control.
  - **Constraints**: N/A.
  - **Impact**: This account will be removed from the `APP_GOVERNOR` role.

Branches and code coverage (including function calls)

Intended branches

- The `account` is removed from the `APP_GOVERNOR` role and an `AppGovernorRemoved` event is emitted.
  ☑ Test coverage

Negative behavior

- The caller does not have the `APP_ROLE_ADMIN` role.
  ☐ Negative test

Function: `remove_app_role_admin(ref self: TState, account: core::starknet::contract_address::ContractAddress)`

This function allows a user with the `GOVERNANCE_ADMIN` role to remove the `APP_ROLE_ADMIN` role from another account and will emit an `AppRoleAdminRemoved` event.

Inputs

- `account`
  - **Control**: Full control.
  - **Constraints**: N/A.
  - **Impact**: This account will be removed from the `APP_ROLE_ADMIN` role.

Branches and code coverage (including function calls)

Intended branches

- The `account` is removed from the `APP_ROLE_ADMIN` role and an `AppRoleAdminRemoved` event is emitted.
  ☑ Test coverage

Negative behavior

- The caller does not have the `GOVERNANCE_ADMIN` role.
  ☐ Negative test

Function: `remove_governance_admin(ref self: TState, account: core::starknet::contract_address::ContractAddress)`

This function allows a user with the `GOVERNANCE_ADMIN` role to remove the `GOVERNANCE_ADMIN` role from another account and will emit a `GovernanceAdminRemoved` event.

Inputs

- `account`
  - **Control**: Full control.
  - **Constraints**: N/A.
  - **Impact**: This account will be removed from the `GOVERNANCE_ADMIN` role.

Branches and code coverage (including function calls)

Intended branches

- The `account` is removed from the `GOVERNANCE_ADMIN` role and a `GovernanceAdminRemoved` event is emitted.

☑ Test coverage

Negative behavior

- The caller does not have the `GOVERNANCE_ADMIN` role.
  ☐ Negative test

Function: `remove_implementation(ref self: TState, implementation_data: src::replaceability_interface::ImplementationData)`

This function removes an implementation from the `impl_activation_time` mapping and will emit a `ImplementationRemoved` event. Only callable by an upgrade governor.

Inputs

- `implementation_data`
  - **Control**: Full control.
  - **Constraints**: N/A.
  - **Impact**: This implementation will no longer be able to be upgraded or downgraded to.

Branches and code coverage

Intended branches

- The implementation is removed from the `impl_activation_time` mapping and an `ImplementationRemoved` event is emitted.
  ☑ Test coverage

Negative behavior

- The caller does not have the `GOVERNANCE_ADMIN` role.
  ☑ Negative test

Function: `remove_operator(ref self: TState, account: core::starknet::contract_address::ContractAddress)`

This function allows a user with the `APP_ROLE_ADMIN` role to remove the `OPERATOR` role from another account and will emit an `OperatorRemoved` event.

Inputs

- `account`
  - **Control**: Full control.
  - **Constraints**: N/A.

- **Impact**: This account will be removed from the `OPERATOR` role.

Branches and code coverage (including function calls)

Intended branches

- The `account` is removed from the `OPERATOR` role and an `OperatorRemoved` event is emitted.
  ☑ Test coverage

Negative behavior

- The caller does not have the `APP_ROLE_ADMIN` role.
  ☐ Negative test

Function: `remove_token_admin(ref self: TState, account: core::starknet::contract_address::ContractAddress)`

This function allows a user with the `APP_ROLE_ADMIN` role to remove the `TOKEN_ADMIN` role from another account and will emit an `TokenAdminRemoved` event.

Inputs

- `account`
  - **Control**: Full control.
  - **Constraints**: N/A.
  - **Impact**: This account will be removed from the `TOKEN_ADMIN` role.

Branches and code coverage (including function calls)

Intended branches

- The `account` is removed from the `TOKEN_ADMIN` role and an `TokenAdminRemoved` event is emitted.
  ☑ Test coverage

Negative behavior

- The caller does not have the `APP_ROLE_ADMIN` role.
  ☐ Negative test

Function: `remove_upgrade_governor(ref self: TState, account: core::starknet::contract_address::ContractAddress)`

This function allows a user with the `GOVERNANCE_ADMIN` role to remove the `UPGRADE_GOVERNOR` role from another account and will emit an `UpgradeGovernorRemoved` event.

Inputs

- `account`
  - **Control**: Full control.
  - **Constraints**: N/A.
  - **Impact**: This account will be removed from the `UPGRADE_GOVERNOR` role.

Branches and code coverage (including function calls)

Intended branches

- The `account` is removed from the `UPGRADE_GOVERNOR` role and an `UpgradeGovernorRemoved` event is emitted.
  ☑ Test coverage

Negative behavior

- The caller does not have the `GOVERNANCE_ADMIN` role.
  ☐ Negative test

Function: `renounce(ref self: TState, role: core::felt252)`

This function allows a user to renounce a role they currently hold and will emit a `RoleRevoked` event.

Inputs

- `role`
  - **Control**: Full control.
  - **Constraints**: Cannot be the `GOVERNANCE_ADMIN` role.
  - **Impact**: This role will be removed from the caller.

Branches and code coverage (including function calls)

Intended branches

- The caller is removed from the role, and a `RoleRevoked` event is emitted.
  ☑ Test coverage

Negative behavior

- The role is `GOVERNANCE_ADMIN`.
  ☑ Negative test

Function: `replace_to(ref self: TState, implementation_data: src::replaceability_interface::ImplementationData)`

This function allows the contract to be upgraded to a new implementation, so long as the activation time has passed for the implementation, the expiration time has not passed, and the contract has not been finalized. Only callable by the upgrade governor.

Inputs

- `implementation_data`
  - **Control**: Full control.
  - **Constraints**: N/A.
  - **Impact**: This contains the class hash that will be used to replace the current contract.

Branches and code coverage (including function calls)

Intended branches

- The contract is upgraded to the specified class hash
  ☑ Test coverage

Negative behavior

- The caller is not the upgrade governor.
  ☑ Negative test
- The activation time has not passed for the implementation.
  ☑ Negative test
- The expiration time has passed for the implementation.
  ☑ Negative test
- The contract has been finalized.
  ☑ Negative test

Function call analysis

- `replace_to` → `eic_initialize(eic_init_data)`
  - **What is controllable?** The `eic_hash` and `eic_init_data` are fully controllable, but must have been specified in the `implementation_data`.

- **If the return value is 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 then the upgrade will be aborted. Since this is a library call, the function will be called with the current state of the contract and can make what ever changes it wants to the storage.

Function: `set_erc20_class_hash(ref self: TState, erc20_class_hash: core::starknet::class_hash::ClassHash)`

This function sets the class hash that will be used when creating new tokens and will emit the `Erc20ClassHashStored` event. Only callable by an app governor.

Inputs

- `erc20_class_hash`
  - **Control**: Full control.
  - **Constraints**: N/A.
  - **Impact**: This class hash will be used for any future token deploys.

Branches and code coverage

Intended branches

- The class hash is set, and the event is emitted.
  ☑ Test coverage

Negative behavior

- The caller is not the app governor.
  ☑ Negative test

Function: `set_l1_bridge(ref self: TState, l1_bridge_address: core::starknet::eth_address::EthAddress)`

This function sets the address of the l1 bridge if it has not already been set, and emits the `L1BridgeSet` event. Only callable by an app governor.

Inputs

- `l1_bridge_address`
  - **Control**: The current `l1_bridge` must be the address zero.
  - **Constraints**: Cannot be address zero.
  - **Impact**: This will be the address that messages are sent to when perform-

ing withdrawals.

Branches and code coverage

Intended branches

- The current `l1_bridge` is address zero, the new address is set, and the `L1BridgeSet` event is emitted.
  - ☑ Test coverage

Negative behavior

- The caller is not an app governor.
  - ☑ Negative test
- The current `l1_bridge` is not address zero.
  - ☑ Negative test
- The new address is address zero.
  - ☑ Negative test

6 Assessment Results

At the time of our assessment, the reviewed code was not deployed.

During our assessment on the scoped Starkgate contracts, we discovered six findings. No critical issues were found. One finding was of medium impact, two were of low impact, and the remaining findings were informational in nature. StarkWare 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 version reviewed during 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. These recommendations are not exhaustive, and we encourage our partners to consider them as a starting point for further discussion. We are happy to provide additional guidance and advice as needed.

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.