Zellic

SushiXSwap V2

Smart Contract Security Assessment

August 22, 2023

Prepared for:

Jiro Ono

SushiSwap

Prepared by:

Filippo Cremonese and Ayaz Mammadov

Zellic Inc.

Contents

About Zellic 3

1 Executive Summary 4

1.1 Goals of the Assessment 4

1.2 Non-goals, Limitations, and Assumptions 4

1.3 Results 5

2 Introduction 6

2.1 About SushiXSwap V2 6

2.2 Methodology 6

2.3 Scope 7

2.4 Project Overview 8

2.5 Project Timeline 8

3 Detailed Findings 9

3.1 USDT transfers can be forced to revert for subsequent users 9

3.2 Refunds sent to `tx.origin` 11

3.3 Gas limit ignored by `executePayload` 12

3.4 Multicall executions with multiple invocations of bridge unintended side effect 13

4 Discussion 15

4.1 Assumptions and intended usage 15

4.2 Axelar Cross chain invariant 16

4.3 Input constraints 17

5 Threat Model 18

5.1 Module: AxelarAdapter.sol 18

5.2 Module: CCTPAdapter.sol 26

5.3 Module: SquidAdapter.sol 33

5.4 Module: StargateAdapter.sol 34

5.5 Module: SushiXSwapV2.sol 41

6 Assessment Results **45**

6.1 Disclaimer 45

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 SushiSwap from July 24th to July 26th, 2023. During this engagement, Zellic reviewed SushiXSwap V2'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:

- Is there any way for a malicious attacker to steal funds from an adapter?
- Do adversarial strategies exist that would allow for griefing or profit from cross-chain transfers (slippage)?
- Are there any edge cases where users could permanently lock up funds?

1.2 Non-goals, Limitations, and Assumptions

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

- Failures in the underlying adapters and bridges used for the cross-chain transfers
- Front-end components
- Infrastructure relating to the project
- Key custody

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

The limited scope and time allocated to this audit prevented us from fully inspecting the interactions between the in-scope contracts and other external contracts, including both contracts owned by Sushi as well as other nontrusted contracts. Due to the high degree of flexibility allowed by the contracts, some potential issues can only be avoided via correct usage. We reviewed the contracts under the assumption that the interactions with them will be mediated by an appropriate front end that does not create potentially vulnerable transactions.

Most importantly, we adopted the assumption that the contracts never hold any asset, except while processing a swap or a cross-chain transfer. Additionally, we assumed that the users do not specify parameters that will invoke malicious contracts while SushiXSwap or an adapter hold any valuable asset. While this assumption is not generally applicable, we believe it is reasonable in the case of SushiXSwap; while the flexibility allowed by the contract might technically allow to use it in a dangerous way, it is our understanding that the front end will prevent dangerous usage.

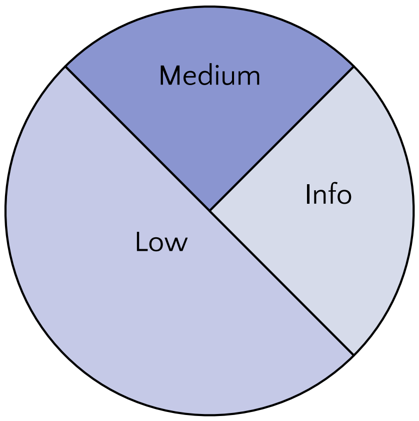
1.3 Results

During our assessment on the scoped SushiXSwap V2 contracts, we discovered four findings. No critical issues were found. One was of medium impact, two were of low impact, and the remaining finding was informational in nature.

Additionally, Zellic recorded its notes and observations from the assessment for SushiSwap'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	1



2 Introduction

2.1 About SushiXSwap V2

SushiXSwap V2 is a revamp of the first version of SushiXSwap. This version was made more flexible with the addition of an adapter that can be added to the main SushiXSwap contract to utilize as many bridges as wanted, and it hooks into the RouteProcessor contract for more efficient swapping across more AMM pool types by updating the RouteProcessor that is used in SushiXSwap.

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 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.

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:

SushiXSwap V2 Contracts

Repository	https://github.com/sushiswap/sushixswap-v2
Version	sushixswap-v2: `6dc20ccc6d03bfed13facff9e7f8692c4e4909bb`
Programs	SushiXSwapV2 AxelarAdapter CCTPAdatper SquidAdapter StargateAdapter AirdropPayloadExecutor

Type	Solidity
Platform	EVM-compatible

2.4 Project Overview

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

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:

Filippo Cremonese, Engineer
fcremo@zellic.io

Ayaz Mammadov, Engineer
ayaz@zellic.io

2.5 Project Timeline

The key dates of the engagement are detailed below.

July 24, 2023	Start of primary review period
July 26, 2023	End of primary review period
August 1, 2023	Draft report delivered

3 Detailed Findings

3.1 USDT transfers can be forced to revert for subsequent users

- **Target**: All Adapters
- **Category**: Business Logic
- **Likelihood**: Medium
- **Severity**: Medium
- **Impact**: Medium

Description

In several adapters, there are external functions designed specifically for use by SushiXSwapV2. However, users can call these functions without restriction, leading to unintended side effects or malicious actions.

```
function swap(
    uint256 _amountBridged,
    bytes calldata _swapData,
    address _token,
    bytes calldata _payloadData
) external payable override {
     ...

    IERC20(rpd.tokenIn).safeIncreaseAllowance(address(rp),
    _amountBridged);

    rp.processRoute(
        rpd.tokenIn,
        _amountBridged ≠ 0 ? _amountBridged : rpd.amountIn,
        rpd.tokenOut,
        rpd.amountOutMin,
        rpd.to,
        rpd.route
    );
}
```

Impact

A malicious user can exploit a specific sequence of function calls to leave an allowance on certain tokens like USDT, which will cause a revert when attempting to approve if the allowance has not been previously set to 0.

An example of this is when a user calls the Axelar adapter's `swap` function and provides a specific `route` to the RouteProcessor that does not fully utilize the allowance. As a result, other users attempting to use the Axelar adapter with USDT will encounter a revert, preventing them from successfully completing the operation.

Recommendations

To fix this issue, it is recommended to zero the USDT allowance where applicable. This will ensure that the allowance is properly reset and prevent the above scenario.

Remediation

This finding was fixed in commit b9f1a4ab by changing from a pull method via allowances to a push method that directly transfers tokens instead.

3.2 Refunds sent to `tx.origin`

- **Target**: AxelarAdapter, CCTPAdapter, StargateAdapter
- **Category**: Coding Mistakes
- **Likelihood**: Low
- **Severity**: Low
- **Impact**: Low

Description

The Axelar, CCTP, and Stargate adapters use `tx.origin` as the address that receives gas refunds in their implementation of `adapterBridge`. This might not be the desired recipient of the refund.

For example, consider the case of an EOA (user) invoking a contract that in turn calls `SushiXSwap` to bridge an asset owned by the contract (and paying for gas using the contract balance). A refund for excess gas would be credited to the user, even though the contract has paid for gas.

Impact

In some cases, gas refunds might be credited to an incorrect recipient.

Recommendations

One possible solution is for `SushiXSwap` to pass the intended recipient for gas refunds to `adapterBridge`. This way, `SushiXSwap` could pass `msg.sender` as the gas refund recipient, which seems to be a more sensible choice.

Remediation

This finding was fixed in commit b9f1a4ab by introducing a variable that allows users to specify an address to refund to.

3.3 Gas limit ignored by `executePayload`

- **Target**: AxelarAdapter, CCTPAdapter, StargateAdapter
- **Category**: Coding Mistakes
- **Severity**: Low
- **Likelihood**: Low
- **Impact**: Low

Description

The Axelar, CCTP, and Stargate adapters' implementation of `executePayload` ignores the `PayloadData::gasLimit` field, which seems to be intended to be used as the gas limit for the call to `PayloadData::target`.

Impact

The target of the `IPayloadExecutor(pd.target).onPayloadReceive` call could use more gas than intended. However, we note that while `executePayload` is an external function, it is intended to be called by `_executeWithToken` or `_execute`, which do limit the gas passed to `executePayload`, preventing the transaction from consuming all the available gas when the contract is used as intended by the developer.

Recommendations

Set the gas limit on the `IPayloadExecutor(pd.target).onPayloadReceive` call to `pd.gasLimit`.

Remediation

This finding was fixed in commit b9f1a4ab by setting the gas limit of the relevant function calls.

3.4 Multicall executions with multiple invocations of bridge unintended side effect

- **Target**: All Adapters
- **Category**: Business Logic
- **Likelihood**: N/A
- **Severity**: Informational
- **Impact**: Informational

Description

When invoking the `bridge` function in SushiXSwapV2, the entire balance is transferred from the contract to the adapter.

```
function bridge(
    BridgeParams calldata _bridgeParams,
    bytes calldata _swapPayload,
    bytes calldata _payloadData
)
    external
    payable
    override
    lock
    onlyApprovedAdapters(_bridgeParams.adapter)
{
     ...
    ISushiXSwapV2Adapter(_bridgeParams.adapter).adapterBridge{
        value: address(this).balance
    }(_bridgeParams.adapterData, _swapPayload, _payloadData);
}
```

Impact

This could be inconvenient for users attempting to multicall the bridge since subsequent calls will have no remaining value left in the contract after the initial transfer to the adapter.

Recommendations

To address this issue, consider adding a parameter to the function that allows users to specify the exact amount they want to send. This way, users can control the amount sent during each call to the function.

Remediation

The SushiSwap team acknowledged this finding and removed multicall from the contract in commit 91acc2a4. They are recommending third parties to consider using `multicall3` if they require batched transactions.

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 Assumptions and intended usage

The contracts in scope for this engagement exhibit a very high degree of flexibility. While no severe inherent vulnerabilities were found, the contracts are only secure if used correctly.

The limited scope of the audit required us to establish several assumptions:

- The contracts never hold any assets, except while processing a swap or bridge transaction; at the end of the transaction, the entire incoming balances are transferred away.
- Users interact directly only with the SushiXSwap contract, using the official front end; they do not interact directly with the adapter contracts.
- The front end (out of scope) prevents dangerous usage of the contracts.
- The contracts do not invoke malicious contracts while holding any valuable assets.

The last assumption is not generally valid, but we believe it to be reasonable and necessary in this context. It is technically possible to use the contracts in a multitude of potentially vulnerable ways. For instance, it may be possible to use the multicall facility to:

- Use `SushiXSwapV2::swap` to pull ERC-20 Token A from `msg.sender`, perform a swap for ERC-20 Token B, and send the output to `AxelarAdapter`.
- Use `SushiXSwapV2::swap` to pull ERC-20 Token C from `msg.sender`, perform a swap for ERC-20 Token D, and send the output to `AxelarAdapter`.
- Use `SushiXSwapV2::bridge` with `tokenIn == NATIVE_ADDRESS`, with `amountIn` equal to the output amount of Token B at Step 1 but `adapterData` encoding `params.token == B` and `params.amount == 0`.
- Use `SushiXSwapV2::bridge` with `tokenIn == NATIVE_ADDRESS`, with `amountIn` equal to the output amount of Token B at Step 1 but `adapterData` encoding `params.token == D` and `params.amount == 0`.

The above sequence of operations would swap A for B, C for D, and then bridge B and D to the intended destination. If Token C or D were malicious, they could perform a cross-contract reentrant call into `AxelarAdapter::executePayload` and steal the `B` tokens. Since `params.amount` is zero, the balance of the token is used in the third step, and the transaction may not revert.

While such usage of the contract is technically possible, we do not believe it will be enabled by the front end.

4.2 Axelar Cross chain invariant

Certain adapters have functionalities that enable them to perform swaps and other actions when receiving tokens. Additionally, they have a quick-exit path that transfers tokens and balances if not enough gas is provided.

```
function _executeWithToken( ... ) internal override {
     ...

    uint256 reserveGas = 100000;

    if (gasleft() < reserveGas) {
        IERC20(_token).safeTransfer(to, amount);

        /// @dev transfer any native token
        if (address(this).balance > 0)
            to.call{value: (address(this).balance)}("");

        return;
    }
     ...
}
```

This retransmission system must be guaranteed to be only be callable by verified relayers otherwise an issue occurs where a malicious user could exploit this feature and intentionally cause a cross-chain swap to fail on the receiving side by reducing the gas supplied. As a consequence, the funds intended for the swap could be sent to a contract that did not expect to receive funds of that type, potentially resulting in the permanent loss of funds.

4.3 Input constraints

The smart contracts in scope could generally apply more input validation and constraints in several cases. Additional validation could give users easier debugging and prevent some potentially damaging mistakes. Some examples follow:

- The `SushiXSwapV2::bridge` does not validate that information contained in `_bridgeParams` top-level fields matches information contained in `_bridgeParams.adapterData`.
  - For instance, `_bridgeParams.tokenIn` does not necessarily match the `token` field of the `AxelarBridgeParams` when `_bridgeParams.adapterData` is decoded as such (when using the Axelar bridge).
- CCTPAdapter assumes to be working with USDC but does not require the `_token` argument to the `executePayload` function to be `nativeUSDC`.
- SushiXSwapV2 does not require that `_bridgeParams.amountIn == address(this).balance` when `_bridgeParams.tokenIn == NATIVE_ADDRESS`; as a result, the `SushiXSwapOnSrc` could contain a misleading amount.

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.

5.1 Module: AxelarAdapter.sol

Function: `adapterBridge(byte[] _adapterData, byte[] _swapData, byte[] _payloadData)`

This function is intended to be called by the SushiXSwapV2 contract to submit a transaction to the cross-chain bridge. However, it can also be called directly.

Inputs

- `_adapterData`
  - **Control**: Arbitrary.
  - **Constraints**: If `params.token == NATIVE_ADDRESS`, `params.amount ≠ 0`.
  - **Impact**: Encodes an `AxelarBridgeParams` struct, which includes the asset to bridge, the amount, the destination chain, and recipient.
- `_swapData`
  - **Control**: Arbitrary.
  - **Constraints**: None.
  - **Impact**: Part of the payload used to invoke the receiver contract on the destination chain.
- `_payloadData`
  - **Control**: Arbitrary.
  - **Constraints**: None.
  - **Impact**: Part of the payload used to invoke the receiver contract on the destination chain.

Branches and code coverage (including function calls)

The function is not tested directly but indirectly by invoking functions on SushiXSwapV2.

Intended branches

- If `token == NATIVE_ADDRESS`, it wraps the native asset into WETH.
  - ☑ Test coverage
- If `params.amount == 0`, it reads the bridged amount from the contract `balanceOf`.
  - ☑ Test coverage
- It grants approval to the gateway for the required amount, pays for gas, and calls the gateway to send a cross-chain call.
  - ☑ Test coverage

Negative behavior

- Reverts if `token == NATIVE_ADDRESS` and `amount == 0`.
  - ☑ Negative test
- Reverts if any of the external calls fail (partially tested for lack of gas payment).
  - ☐ Negative test

Function call analysis

- `rootFunction → weth.deposit{value: params.amount}()`
  - **What is controllable?** `params.amount`.
  - **If return value controllable, how is it used and how can it go wrong?** Not used.
  - **What happens if it reverts, reenters, or does other unusual control flow?** Reverts nor reentrancy are possible with standard WETH.
- `rootFunction → IERC20(params.token).balanceOf(address(this))`
  - **What is controllable?** `params.token`.
  - **If return value controllable, how is it used and how can it go wrong?** Used as `params.amount`.
  - **What happens if it reverts, reenters, or does other unusual control flow?** Reverts bubble up; reentrancy is possible but not an issue if the contract is used as intended.
- `rootFunction → IERC20(params.token).safeApprove(address(gateway), params.amount)`
  - **What is controllable?** `params.token` and `params.amount`.
  - **If return value controllable, how is it used and how can it go wrong?** Not used.
  - **What happens if it reverts, reenters, or does other unusual control flow?** Reverts bubble up; reentrancy is possible but not an issue if the contract is used as intended.
- `rootFunction → axelarGasService.payNativeGasForContractCallWithToken(...)`

  - **What is controllable?** All arguments derived from `params` and `payload`.
  - **If return value controllable, how is it used and how can it go wrong?** Not used.
  - **What happens if it reverts, reenters, or does other unusual control flow?** Reverts bubble up; reentrancy is possible but not an issue if the contract is used as intended.
- `rootFunction` → `gateway.callContractWithToken( ... )`
  - **What is controllable?** All arguments.
  - **If return value controllable, how is it used and how can it go wrong?** Not used.
  - **What happens if it reverts, reenters, or does other unusual control flow?** Reverts bubble up; reentrancy is possible but not an issue if the contract is used as intended.

Function: `executePayload(uint256 _amountBridged, byte[] _payloadData, address _token)`

This function is intended to be invoked by `_executeWithToken` on the destination chain to transfer the received asset to a target contract and invoke a callback on it. It can also be invoked directly.

Inputs

- `_amountBridged`
  - **Control**: Arbitrary.
  - **Constraints**: None.
  - **Impact**: Represents the amount of bridged asset.
- `_payloadData`
  - **Control**: Arbitrary.
  - **Constraints**: None.
  - **Impact**: Encodes the target that receives the bridged asset as well as the payload for the `IPayloadExecutor` callback.
- `_token`
  - **Control**: Arbitrary.
  - **Constraints**: None.
  - **Impact**: Address of the bridged asset.

Branches and code coverage (including function calls)

The function is tested indirectly by invoking `_executeWithToken`.

Intended branches

- Transfers the `_token` to the target and invokes `onPayloadReceive` on the target
  - ☑ Test coverage

Negative behavior

- Reverts if the target callback reverts.
  - ☐ Negative test

Function call analysis

- `rootFunction` → `IERC20(_token).safeTransfer(pd.target, _amountBridged)`
  - **What is controllable?** `_token`, `pd.target`, and `_amountBridged`.
  - **If return value controllable, how is it used and how can it go wrong?** Not used.
  - **What happens if it reverts, reenters, or does other unusual control flow?** Reverts bubble up; reentrancy is possible but not an issue if the contract is used correctly.
- `rootFunction` → `IPayloadExecutor(pd.target).onPayloadReceive(pd.targetData)`
  - **What is controllable?** `pd.target`, `pd.targetData`.
  - **If return value controllable, how is it used and how can it go wrong?** Not used.
  - **What happens if it reverts, reenters, or does other unusual control flow?** Reverts bubble up; reentrancy is possible but not an issue if the contract is used correctly.

Function: `swap(uint256 _amountBridged, byte[] _swapData, address _token, byte[] _payloadData)`

This function is intended to be invoked by `_executeWithToken` on the destination chain to complete a swap. It can also be invoked directly.

Inputs

- `_amountBridged`
  - **Control**: Arbitrary.
  - **Constraints**: None.
  - **Impact**: Represents the amount of the bridged asset.
- `_swapData`
  - **Control**: Arbitrary.
  - **Constraints**: None (must encode a `RouteProcessorData` struct).

    - **Impact**: Encodes route processor data and identifies in/out assets, input amount, minimum output amount, route, and recipient.
- `_token`
    - **Control**: Arbitrary.
    - **Constraints**: None.
    - **Impact**: Not used.
- `_payloadData`
    - **Control**: Arbitrary.
    - **Constraints**: None.
    - **Impact**: Address and payload for the `IPayloadExecutor.onPayloadReceive` callback.

Branches and code coverage (including function calls)

Intended branches

- Grants approval to the RP for the received asset, invokes the RP, and invokes the `IPayloadExecutor` callback only if `_payloadData` was supplied.
    ☑ Test coverage

Negative behavior

- Reverts if `_swapData` is invalid.
    ☐ Negative test
- Reverts if the route processor reverts.
    ☐ Negative test
- Reverts if the `IPayloadExecutor` callback reverts.
    ☐ Negative test

Function call analysis

- `rootFunction → IERC20(rpd.tokenIn).safeIncreaseAllowance( ... )`
    - **What is controllable?** `tokenIn` and `_amountBridged`.
    - **If return value controllable, how is it used and how can it go wrong?** Not used.
    - **What happens if it reverts, reenters, or does other unusual control flow?** Reverts bubble up; reentrancy is possible but not an issue as long as the contract is used as intended.
- `rootFunction → rp.processRoute( ... )`
    - **What is controllable?** All arguments.
    - **If return value controllable, how is it used and how can it go wrong?** Not used.

- **What happens if it reverts, reenters, or does other unusual control flow?** Reverts bubble up; reentrancy is possible but not an issue as long as the contract is used as intended.

- `rootFunction` → `IPayloadExecutor(pd.target).onPayloadReceive(pd.targetData)`
  - **What is controllable?** `target` and `targetData`.
  - **If return value controllable, how is it used and how can it go wrong?** Not used.
  - **What happens if it reverts, reenters, or does other unusual control flow?** Reverts are caught, and another revert is issued. Reentrancy is possible but not an issue as long as the contract is used as intended.

Function: `_executeWithToken(string memory sourceChain, string memory sourceAddress, bytes calldata payload, string memory tokenSymbol, uint256 amount)`

This internal function is invoked by the inherited external function `executeWithToken`, which validates that a cross-chain function call is legitimate and not repeated.

Inputs

- `sourceChain`
  - **Control**: None (determined by the source chain).
  - **Constraints**: None.
  - **Impact**: Unused.
- `sourceAddress`
  - **Control**: None (corresponds to the actual source address).
  - **Constraints**: None.
  - **Impact**: Unused.
- `payload`
  - **Control**: Arbitrary.
  - **Constraints**: None (apart from encoding a valid `address, bytes, bytes` tuple).
  - **Impact**: Determines the values of the `to`, `_swapData`, and `_payloadData` variables.
- `tokenSymbol`
  - **Control**: None.
  - **Constraints**: None.
  - **Impact**: Used to get the `_token` address from the gateway.
- `amount`

  - **Control**: None (corresponds to the bridged amount).
  - **Constraints**: None.
  - **Impact**: Determines the bridged amount.

Branches and code coverage (including function calls)

Intended branches

- If the gas left is lower than the minimum `reserveGas`, it transfers the `_token` amount and the entire contract balance to the `to` address and returns early.
  ☑ Test coverage
- If `gas > reserveGas` and `_swapData.length > 0`, it performs a swap, then transfers the remaining `_token` and native asset balance to the `to` address.
  ☑ Test coverage
- If `gas > reserveGas`, `_swapData.length == 0` and `_payloadData.length > 0`, it performs a callback with `executePayload`, then transfers the remaining `_token` and native asset balance to the `to` address.
  ☑ Test coverage
- If `gas > reserveGas`, `_swapData.length == 0` and `_payloadData.length == 0`, just transfers the remaining `_token` and native asset balance to the `to` address.
  ☑ Test coverage

Negative behavior

- Correctly handles a failed swap; without reverting, it transfers the received assets to the recipient (`to` address).
  ☑ Negative test
- Correctly handles a failed callback; without reverting, it transfers the received assets to the recipient (`to` address).
  ☑ Negative test

Function call analysis

For all the following calls, in a normal context of operation, uncaught reverts imply a failure to transfer the bridged assets and likely these assets becoming (at least temporarily) stuck.

- `rootFunction` → `gateway.tokenAddresses(tokenSymbol)`
  - **What is controllable?** `tokenSymbol`.
  - **If return value controllable, how is it used and how can it go wrong?** Used as the `_token` address.
  - **What happens if it reverts, reenters, or does other unusual control flow?** Reverts bubble up; reentrancy is considered not possible (`gateway` is as-

sumed to be trusted).

- `rootFunction` → `IERC20(_token).safeTransfer(to, amount)`
  - **What is controllable?** `_token`, `to`, and `amount`.
  - **If return value controllable, how is it used and how can it go wrong?** Not used.
  - **What happens if it reverts, reenters, or does other unusual control flow?** Reverts bubble up; reentrancy is possible but not an issue if the contract is used as intended.
- `rootFunction` → `to.call{value: (address(this).balance)}(“”)`
  - **What is controllable?** `to`.
  - **If return value controllable, how is it used and how can it go wrong?** Not used.
  - **What happens if it reverts, reenters, or does other unusual control flow?** Reverts bubble up; reentrancy is possible but not an issue if the contract is used as intended.
- `rootFunction` → `ISushiXSwapV2Adapter(address(this)).swap( … )`
  - **What is controllable?** `amount`, `_swapData`, `_token`, and `_payloadData`.
  - **If return value controllable, how is it used and how can it go wrong?** Not used.
  - **What happens if it reverts, reenters, or does other unusual control flow?** Reverts are caught and ignored (on purpose). Reentrancy is possible but not an issue if the contract is used as intended.
- `rootFunction` → `ISushiXSwapV2Adapter(address(this)).executePayload( … )`
  - **What is controllable?** `amount`, `_payloadData`, and `_token`.
  - **If return value controllable, how is it used and how can it go wrong?** Not used.
  - **What happens if it reverts, reenters, or does other unusual control flow?** Reverts are caught and ignored (on purpose). Reentrancy is possible but not an issue if the contract is used as intended.
- `rootFunction` → `IERC20(_token).balanceOf(address(this)`
  - **What is controllable?** `_token`.
  - **If return value controllable, how is it used and how can it go wrong?** Used to determine whether there is a remaining token balance that needs to be transferred.
  - **What happens if it reverts, reenters, or does other unusual control flow?** Reverts bubble up; reentrancy is possible but not an issue if the contract is used as intended.
- `rootFunction` → `IERC20(_token).safeTransfer(to, amount)`
  - **What is controllable?** `_token`, `to`, and `amount`.
  - **If return value controllable, how is it used and how can it go wrong?** Not

used.
    - **What happens if it reverts, reenters, or does other unusual control flow?** Reverts bubble up; reentrancy is possible but not an issue if the contract is used as intended.
- `rootFunction → to.call{value: (address(this).balance)}(“”)`
    - **What is controllable?** `to`.
    - **If return value controllable, how is it used and how can it go wrong?** Not used.
    - **What happens if it reverts, reenters, or does other unusual control flow?** Reverts bubble up; reentrancy is possible but not an issue if the contract is used as intended.

5.2 Module: CCTPAdapter.sol

Function: `adapterBridge(byte[] _adapterData, byte[] _payloadData)`

This function is intended to be called by the SushiXSwapV2 contract to submit a transaction to the cross-chain bridge. However, it can also be called directly.

Inputs

- `_adapterData`
    - **Control**: Arbitrary.
    - **Constraints**: If `params.token == NATIVE_ADDRESS`, `params.amount ≠ 0`.
    - **Impact**: Encodes an `AxelarBridgeParams` struct, which includes the asset to bridge, the amount, the destination chain, and recipient.
- `_swapData`
    - **Control**: Arbitrary.
    - **Constraints**: None.
    - **Impact**: Part of the payload used to invoke the receiver contract on the destination chain.
- `_payloadData`
    - **Control**: Arbitrary.
    - **Constraints**: None.
    - **Impact**: Part of the payload used to invoke the receiver contract on the destination chain.

Branches and code coverage (including function calls)

Intended branches

- If `params.amount == 0`, it uses the USDC balance as `params.amount`.
  - ☑ Test coverage
- Approves `tokenMessenger` for the bridged amount, burns the USDC to bridge it to the destination, pays for gas using the Axelar gas service, and invokes the Axelar gateway to initiate cross-chain calls.
  - ☑ Test coverage

Negative behavior

- Reverts if the contract USDC balance is zero.
  - ☑ Negative test
- Reverts if any of the external calls fail (partially tested for lack of gas payment).
  - ☐ Negative test

Function call analysis

- `rootFunction → nativeUSDC.balanceOf(address(this))`
  - **What is controllable?** Nothing.
  - **If return value controllable, how is it used and how can it go wrong?** Used to ensure contract USDC balance is greater than zero and to set `params.amount`.
  - **What happens if it reverts, reenters, or does other unusual control flow?** Reverts and reentrancy are not possible.
- `rootFunction → nativeUSDC.safeApprove(address(tokenMessenger), params.amount)`
  - **What is controllable?** `params.amount`.
  - **If return value controllable, how is it used and how can it go wrong?** Not used.
  - **What happens if it reverts, reenters, or does other unusual control flow?** Reverts and reentrancy are not possible.
- `rootFunction → tokenMessenger.depositForBurn( … )`
  - **What is controllable?** All arguments except `burnToken`.
  - **If return value controllable, how is it used and how can it go wrong?** Not used.
  - **What happens if it reverts, reenters, or does other unusual control flow?** Reverts bubble up; reentrancy is not possible.
- `rootFunction → axelarGasService.payNativeGasForContractCall( … )`
  - **What is controllable?** Arguments derived from `params` and `payload`.

  - **If return value controllable, how is it used and how can it go wrong?** Not used.
  - **What happens if it reverts, reenters, or does other unusual control flow?** Reverts bubble up; reentrancy should not be possible and is not an issue if the contract is used correctly.
- `rootFunction` → `gateway.callContract( ... )`
  - **What is controllable?** All arguments.
  - **If return value controllable, how is it used and how can it go wrong?** Not used.
  - **What happens if it reverts, reenters, or does other unusual control flow?** Reverts bubble up; reentrancy should not be possible and is not an issue if the contract is used correctly.

Function: `executePayload(uint256 _amountBridged, byte[] _payloadData, address _token)`

This function is intended to be invoked by `_executeWithToken` on the destination chain to transfer the received USDC to a target contract and invoke a callback on it. It can also be invoked directly.

Inputs

- `_amountBridged`
  - **Control**: Arbitrary.
  - **Constraints**: None.
  - **Impact**: Represents the amount of bridged asset.
- `_payloadData`
  - **Control**: Arbitrary.
  - **Constraints**: None.
  - **Impact**: Encodes the target that receives the bridged asset as well as the payload for the `IPayloadExecutor` callback.
- `_token`
  - **Control**: Arbitrary.
  - **Constraints**: None.
  - **Impact**: Unused.

Branches and code coverage (including function calls)

The function is tested indirectly by invoking `_executeWithToken`.

Intended branches

- Transfers the `_token` to the target and invokes `onPayloadReceive` on the target.
  - ☑ Test coverage

Negative behavior

- Reverts if the target callback reverts.
  - ☐ Negative test

Function call analysis

- `rootFunction` → `IERC20(_token).safeTransfer(pd.target, _amountBridged)`
  - **What is controllable?** `_token`, `pd.target`, and `_amountBridged`.
  - **If return value controllable, how is it used and how can it go wrong?** Not used.
  - **What happens if it reverts, reenters, or does other unusual control flow?** Reverts bubble up; reentrancy is possible but not an issue if the contract is used correctly.
- `rootFunction` → `IPayloadExecutor(pd.target).onPayloadReceive(pd.targetData)`
  - **What is controllable?** `pd.target` and `pd.targetData`.
  - **If return value controllable, how is it used and how can it go wrong?** Not used.
  - **What happens if it reverts, reenters, or does other unusual control flow?** Reverts bubble up; reentrancy is possible but not an issue if the contract is used correctly.

Function: `swap(uint256 _amountBridged, byte[] _swapData, address _token, byte[] _payloadData)`

This function is intended to be invoked by `_executeWithToken` on the destination chain to complete a swap. It can also be invoked directly.

Inputs

- `_amountBridged`
  - **Control**: Arbitrary.
  - **Constraints**: None.
  - **Impact**: Represents the amount of the bridged asset.
- `_swapData`
  - **Control**: Arbitrary.
  - **Constraints**: None (must encode a `RouteProcessorData` struct).
  - **Impact**: Encodes route processor data and identifies in/out assets, input

amount, minimum output amount, route, and recipient.

- `_token`
  - **Control**: Arbitrary.
  - **Constraints**: None.
  - **Impact**: Not used.
- `_payloadData`
  - **Control**: Arbitrary.
  - **Constraints**: None.
  - **Impact**: Address and payload for the `IPayloadExecutor.onPayloadReceive` callback.

Branches and code coverage (including function calls)

Intended branches

- Grants approval to the RP for the received asset, invokes the RP, and invokes the `IPayloadExecutor` callback only if `_payloadData` was supplied.
  - ☑ Test coverage

Negative behavior

- Reverts if `_swapData` is invalid.
  - ☐ Negative test
- Reverts if the route processor reverts.
  - ☐ Negative test
- Reverts if the `IPayloadExecutor` callback reverts.
  - ☐ Negative test

Function call analysis

- `rootFunction` → `IERC20(rpd.tokenIn).safeIncreaseAllowance( ... )`
  - **What is controllable?** `tokenIn` and `_amountBridged`.
  - **If return value controllable, how is it used and how can it go wrong?** Not used.
  - **What happens if it reverts, reenters, or does other unusual control flow?** Reverts bubble up; reentrancy is possible but not an issue as long as the contract is used as intended.
- `rootFunction` → `rp.processRoute( ... )`
  - **What is controllable?** All arguments.
  - **If return value controllable, how is it used and how can it go wrong?** Not used.
  - **What happens if it reverts, reenters, or does other unusual control flow?**

Reverts bubble up; reentrancy is possible but not an issue as long as the contract is used as intended.

- `rootFunction` → `IPayloadExecutor(pd.target).onPayloadReceive(pd.targetData)`
  - **What is controllable?** `target` and `targetData`.
  - **If return value controllable, how is it used and how can it go wrong?** Not used.
  - **What happens if it reverts, reenters, or does other unusual control flow?** Reverts are caught, and another revert is issued. Reentrancy is possible but not an issue as long as the contract is used as intended.

Function: `_execute(string memory, string memory, bytes calldata payload)`

This internal function is invoked by the inherited `invoke` external function.

Inputs

- Argument 1
  - **Control**: None (corresponds to the source chain).
  - **Constraints**: None.
  - **Impact**: Not used.
- Argument 2
  - **Control**: None (corresponds to the sender address).
  - **Constraints**: None.
  - **Impact**: Not used.
- `payload`
  - **Control**: Arbitrary.
  - **Constraints**: Must encode an `address, uint256, bytes, bytes` tuple.
  - **Impact**: Encodes `to`, `amount`, `_swapData`, and `_payloadData`.

Branches and code coverage (including function calls)

Intended branches

- If gas left is less than `reserveGas`, it transfers the remaining asset balances and returns early.
  - ☑ Test coverage
- Performs a call to `swap` if `_swapData` is not empty.
  - ☑ Test coverage
- Performs a call to `executePayload` if `_swapData` is empty and `_payloadData` is not.

  - ☑ Test coverage
- Transfers the remaining asset balances to the `to` address.
  - ☑ Test coverage

Negative behavior

- Correctly handles a failed swap; without reverting, it transfers the received assets to the recipient (`to` address).
  - ☑ Negative test
- Correctly handles a failed callback; without reverting, it transfers the received assets to the recipient (`to` address).
  - ☑ Negative test

Function call analysis

- `rootFunction → nativeUSDC.safeTransfer(to, amount)`
  - **What is controllable?** `to` and `amount`.
  - **If return value controllable, how is it used and how can it go wrong?** Not used.
  - **What happens if it reverts, reenters, or does other unusual control flow?** Reverts bubble up; reentrancy is not possible.
- `rootFunction → to.call{value: (address(this).balance)}(“”)`
  - **What is controllable?** `to`.
  - **If return value controllable, how is it used and how can it go wrong?** Not used.
  - **What happens if it reverts, reenters, or does other unusual control flow?** Reverts bubble up; reentrancy is possible but not an issue if the contract is used as intended.
- `rootFunction → ISushiXSwapV2Adapter(address(this)).swap( … )`
  - **What is controllable?** `amount`, `_swapData`, and `_payloadData`.
  - **If return value controllable, how is it used and how can it go wrong?** Not used.
  - **What happens if it reverts, reenters, or does other unusual control flow?** Reverts are caught and ignored (on purpose). Reentrancy is possible, but not an issue if the contract is used as intended.
- `rootFunction → ISushiXSwapV2Adapter(address(this)).executePayload( … )`
  - **What is controllable?** `amount` and `_payloadData`.
  - **If return value controllable, how is it used and how can it go wrong?** Not used.
  - **What happens if it reverts, reenters, or does other unusual control flow?** Reverts are caught and ignored (on purpose). Reentrancy is possible but

not an issue if the contract is used as intended.

- `rootFunction → nativeUSDC.balanceOf(address(this)`
  - **What is controllable?** Nothing.
  - **If return value controllable, how is it used and how can it go wrong?** Used to determine whether there is a remaining USDC balance that needs to be transferred.
  - **What happens if it reverts, reenters, or does other unusual control flow?** Reverts and reentrancy are not possible.
- `rootFunction → nativeUSDC.safeTransfer(to, amount)`
  - **What is controllable?** `to` and `amount`.
  - **If return value controllable, how is it used and how can it go wrong?** Not used.
  - **What happens if it reverts, reenters, or does other unusual control flow?** Reverts bubble up; reentrancy is not possible.
- `rootFunction → to.call{value: (address(this).balance)}("")`
  - **What is controllable?** `to`.
  - **If return value controllable, how is it used and how can it go wrong?** Not used.
  - **What happens if it reverts, reenters, or does other unusual control flow?** Reverts bubble up; reentrancy is possible but not an issue if the contract is used as intended.

5.3 Module: SquidAdapter.sol

Function: `adapterBridge(byte[] _adapterData, byte[] None, byte[] None)`

This is an adapter interface for handling this bridge.

Inputs

- `_adapterData`
  - **Control**: `adapterData`.
  - **Constraints**: None.
  - **Impact**: Data specific to this adapter (decoded into the token and the `squidRouterdata`).

Branches and code coverage (including function calls)

Intended branches

- Calls the `squidRouter`.

    - ☑ Test coverage
- Handles the native token.
    - ☑ Test coverage

Function call analysis

- `IERC20(token).safeApprove( … )`
    - **What is controllable?** Nothing.
    - **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?** USDT needs zero approval.
- `IERC20(token).balanceOf(this)`
    - **What is controllable?** Nothing.
    - **If return value controllable, how is it used and how can it go wrong?** The amount of tokens held.
    - **What happens if it reverts, reenters, or does other unusual control flow?** Nothing.
- `IERC20(token).balanceOf(this)`
    - **What is controllable?** Nothing.
    - **If return value controllable, how is it used and how can it go wrong?** The amount of tokens held.
    - **What happens if it reverts, reenters, or does other unusual control flow?** Nothing.

5.4 Module: StargateAdapter.sol

Function: `adapterBridge(byte[] _adapterData, byte[] _swapData, byte[] _payloadData)`

This is the adapter bridge interface.

Inputs

- `_adapterData`
    - **Control**: Full.
    - **Constraints**: None.
    - **Impact**: Data specific for this adapter.
- `_swapData`
    - **Control**: Full.

  - **Constraints**: None.
  - **Impact**: SG router data.
- `_payloadData`
  - **Control**: Full.
  - **Constraints**: None.
  - **Impact**: Payload.

Branches and code coverage (including function calls)

Intended branches

- Handles native tokens correctly.
  - ☑ Test coverage
- Handles WETH correctly.
  - ☑ Test coverage

Function call analysis

- `IStargateEthVault(sgeth).deposit{value: params.amount}()`
  - **What is controllable?** Everything.
  - **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?** Nothing.
- `weth.balanceOf(address(this))`
  - **What is controllable?** Everything.
  - **If return value controllable, how is it used and how can it go wrong?** WETH balance.
  - **What happens if it reverts, reenters, or does other unusual control flow?** Nothing.
- `weth.withdraw(params.amount)`
  - **What is controllable?** Everything.
  - **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?** Nothing.
- `IERC20(params.token).balanceOf(address(this))`
  - **What is controllable?** Everything.
  - **If return value controllable, how is it used and how can it go wrong?** Balance of token of this.
  - **What happens if it reverts, reenters, or does other unusual control flow?**

Nothing.

- `IERC20(params.token).safeApprove( address(stargateRouter), params.amount );`
  - **What is controllable?** Params.
  - **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?** Nothing.
- `stargateRouter.swap{value: address(this).balance}( … )`
  - **What is controllable?** Everything.
  - **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?** Nothing.

Function: `executePayload(uint256 _amountBridged, byte[] _payloadData, address _token)`

This executes payload.

Inputs

- `_amountBridged`
  - **Control**: Full.
  - **Constraints**: None.
  - **Impact**: Amount bridged.
- `_payloadData`
  - **Control**: Full.
  - **Constraints**: None.
  - **Impact**: Payload data for the target.
- `_token`
  - **Control**: Full.
  - **Constraints**: None.
  - **Impact**: The token.

Branches and code coverage (including function calls)

Intended branches

- Handles special/native tokens.
  - ☑ Test coverage

Negative behavior

- Reverts if the target contract reverts
  - ☐ Test coverage

Function call analysis

- `weth.deposit{value: _amountBridged}()`
  - **What is controllable?** Everything.
  - **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?** Nothing.
- `IERC20(_token).safeTransfer(pd.target, _amountBridged)`
  - **What is controllable?** Everything.
  - **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?** Nothing.
- `IPayloadExecutor(pd.target).onPayloadReceive(pd.targetData)`
  - **What is controllable?** Everything.
  - **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?** Nothing.

Function: `sgReceive(uint16 None, byte[] None, uint256 None, address _token, uint256 amountLD, byte[] payload)`

The receiving side of the adapter.

Inputs

- `_token`
  - **Control**: Full.
  - **Constraints**: None.
  - **Impact**: The token.
- `amountLD`
  - **Control**: Full.
  - **Constraints**: None.
  - **Impact**: Amount.

- `payload`
  - **Control**: Full.
  - **Constraints**: None.
  - **Impact**: Received payload.

Branches and code coverage (including function calls)

Intended branches

- Transfers and receives the payload correctly.
  - ☑ Test coverage

Negative behavior

- Can only be called by the bridge.
  - ☐ Negative test

Function call analysis

- `IERC20(_token).safeTransfer(to, amountLD)`
  - **What is controllable?** `to` and `amountLD`.
  - **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?** Nothing.
- `to.call{value: (address(this).balance)}("")`
  - **What is controllable?** Nothing.
  - **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?** Nothing.
- `ISushiXSwapV2Adapter(address(this)).swap{gas: limit}( amountLD, _swapData, _token, _payloadData)`
  - **What is controllable?** Everything.
  - **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?** Nothing.
- `ISushiXSwapV2Adapter(address(this)).swap{gas: limit}( amountLD, _swapData, _token, _payloadData)`
  - **What is controllable?** Everything.
  - **If return value controllable, how is it used and how can it go wrong?** Dis-

carded.
  - **What happens if it reverts, reenters, or does other unusual control flow?** Nothing.
- `ISushiXSwapV2Adapter(address(this)).executePayload{gas: limit}( amountLD , _payloadData, _token)`
  - **What is controllable?** Everything.
  - **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?** Nothing.
- `IERC20(_token).balanceOf(address(this))`
  - **What is controllable?** `_token`.
  - **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?** Nothing.
- `IERC20(_token).safeTransfer(to, amountLD)`
  - **What is controllable?** `_token`.
  - **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?** Nothing.

Function: `swap(uint256 _amountBridged, byte[] _swapData, address _token , byte[] _payloadData)`

This swaps functionality.

Inputs

- `_amountBridged`
  - **Control**: Full.
  - **Constraints**: None.
  - **Impact**: Amount to bridge.
- `_swapData`
  - **Control**: Full.
  - **Constraints**: None.
  - **Impact**: Data for the swap functionality.
- `_token`
  - **Control**: Full.

  - **Constraints**: None.
  - **Impact**: Token.
- `_payloadData`
  - **Control**: Full.
  - **Constraints**: None.
  - **Impact**: Payload for the PD target.

Branches and code coverage (including function calls)

Intended branches

- Swaps the tokens.
  ☑ Test coverage
- Respects the minimum amount out.
  ☑ Test coverage
- Handles native and special tokens (SGETH).
  ☑ Test coverage

Function call analysis

- `IERC20(rpd.tokenIn).safeIncreaseAllowance(address(rp), _amountBridged)`
  - **What is controllable?** `tokenIn` and `amountBridged`.
  - **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?** Nothing.
- `rp.processRoute( rpd.tokenIn, _amountBridged ≠ 0 ? _amountBridged : rpd.amountIn, rpd.tokenOut, rpd.amountOutMin, rpd.to, rpd.route)`
  - **What is controllable?** Everything.
  - **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?** Reentrancy and unexpected effects if the route is not correct.
- `IPayloadExecutor(pd.target).onPayloadReceive(pd.targetData)`
  - **What is controllable?** Everything.
  - **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?** Nothing (try/catch).

5.5 Module: SushiXSwapV2.sol

Function: `bridge(BridgeParams _bridgeParams, byte[] _swapPayload, byte[] _payloadData)`

This function can be used to bridge a token using any of the allowed adapters.

Inputs

- `_bridgeParams`
  - **Control**: Arbitrary.
  - **Constraints**: `onlyApprovedAdapters(_bridgeParams.adapter)`.
  - **Impact**: Determines the adapter to use, the token and amount to bridge, and the recipient.
- `_swapPayload`
  - **Control**: Arbitrary.
  - **Constraints**: None.
  - **Impact**: Bytes payload passed verbatim to the adapter.
- `_payloadData`
  - **Control**: Arbitrary.
  - **Constraints**: None.
  - **Impact**: Bytes payload passed verbatim to the adapter.

Branches and code coverage (including function calls)

Intended branches

- If transferring an ERC-20 token, it transfers the required amount from the caller to the adapter, then invokes the adapter.
  - [x] Test coverage
- If transferring ETH, it directly invokes the adapter (transferring the contract ETH balance).
  - [x] Test coverage
- Tests for all individual adapters.
  - [ ] Test coverage

Negative behavior

- Reverts if the adapter is not approved.
  - [ ] Negative test
- Reverts if the contract is locked.
  - [ ] Negative test

- Reverts if the adapter reverts due to, for example, unsupported asset, insufficient gas, and so forth (tested for some individual adapters).
  - ☑ Negative test

Function call analysis

- `rootFunction` → `IERC20(_bridgeParams.tokenIn).safeTransferFrom( … )`
  - **What is controllable?** `tokenIn`, `adapter`, and `amountIn`.
  - **If return value controllable, how is it used and how can it go wrong?** Not used.
  - **What happens if it reverts, reenters, or does other unusual control flow?** Reverts bubble up. Reentrancy is prevented with guards, which do not cover the separate adapter contracts, however.
- `rootFunction` → `ISushiXSwapV2Adapter(_bridgeParams.adapter).adapterBridge( … )`
  - **What is controllable?** `adapter`, `adapterData`, `_swapPayload`, and `_payloadData`.
  - **If return value controllable, how is it used and how can it go wrong?** Not used.
  - **What happens if it reverts, reenters, or does other unusual control flow?** Reverts bubble up. Reentrancy is prevented with guards, which do not cover the separate adapter contracts, however.

Function: `swapAndBridge(BridgeParams _bridgeParams, byte[] _swapData, byte[] _swapPayload, byte[] _payloadData)`

This swaps and then calls bridge.

Inputs

- `_bridgeParams`
  - **Control**: Full.
  - **Constraints**: None.
  - **Impact**: Which bridge to use.
- `_swapData`
  - **Control**: Full.
  - **Constraints**: None.
  - **Impact**: The data used for the swap.
- `_swapPayload`
  - **Control**: Full.
  - **Constraints**: None.

    - **Impact**: Swaps payload for the bridge adapter.
- `_payloadData`
    - **Control**: Full.
    - **Constraints**: None.
    - **Impact**: Payload for the bridge.

Branches and code coverage (including function calls)

Intended branches

- Handles native tokens and transfers balance.
  ☑ Test coverage
- Performs a swap if the appropriate swap data is provided.
  ☑ Test coverage
- Calls the bridge adapter.
  ☐ Test coverage

Negative behavior

- Cannot use an unapproved bridge.
  ☑ Negative test

Function call analysis

- `ISushiXSwapV2Adapter(_bridgeParams.adapter).adapterBridge{ value: address(this).balance }(_bridgeParams.adapterData, _swapPayload, _payloadData)`
    - **What is controllable?** Everything.
    - **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?** Nothing.

Function: `swap(byte[] _swapData)`

This swaps.

Inputs

- `_swapData`
    - **Control**: Full.
    - **Constraints**: None.
    - **Impact**: The swap data.

Branches and code coverage (including function calls)

Intended branches

- Swaps as intended.
  - ☑ Test coverage
- Handles the native token.
  - ☑ Test coverage

Negative behaviour

Function call analysis

- `_swap( ... )` → `IERC20(rpd.tokenIn).safeTransferFrom( msg.sender, address( this), rpd.amountIn)`
  - **What is controllable?** RPD.
  - **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?** Nothing.
- `_swap( ... )` → `weth.deposit{value: rpd.amountIn}()`
  - **What is controllable?** RPD.
  - **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?** Nothing.
- `_swap( ... )` → `IERC20(rpd.tokenIn).safeIncreaseAllowance(address(rp), rpd.amountIn)`
  - **What is controllable?** RPD.
  - **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?** Nothing.
- `_swap( ... )` → `rp.processRoute( rpd.tokenIn, rpd.amountIn, rpd.tokenOut, rpd.amountOutMin, rpd.to, rpd.route)`
  - **What is controllable?** RPD.
  - **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?** Nothing.

6 Assessment Results

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

During our assessment on the scoped SushiXSwap V2 contracts, we discovered four findings. No critical issues were found. One was of medium impact, two were of low impact, and the remaining finding was informational in nature. SushiSwap 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.